pomegranate

fast and flexible probabilistic modelling in python

Jacob Schreiber
Paul G. Allen School of Computer Science
University of Washington

 jmschreiber91

 @jmschrei

 @jmschreiber91

Acknowledgements

UNIVERSITY of WASHINGTON
eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS

Inria
INVENTORS FOR THE DIGITAL WORLD

aspentech

Overview

pomegranate is more flexible than other packages, faster, is intuitive to use, and can do it all in parallel

Overview: this talk

Overview

Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

Overview: supported models

Six Main Models:

1. Probability Distributions
2. General Mixture Models
3. Markov Chains
4. Hidden Markov Models
5. Bayes Classifiers / Naive Bayes
6. Bayesian Networks

Two Helper Models:

1. k-means++/kmeans||
2. Factor Graphs

Overview: model stacking in pomegranate

Distributions

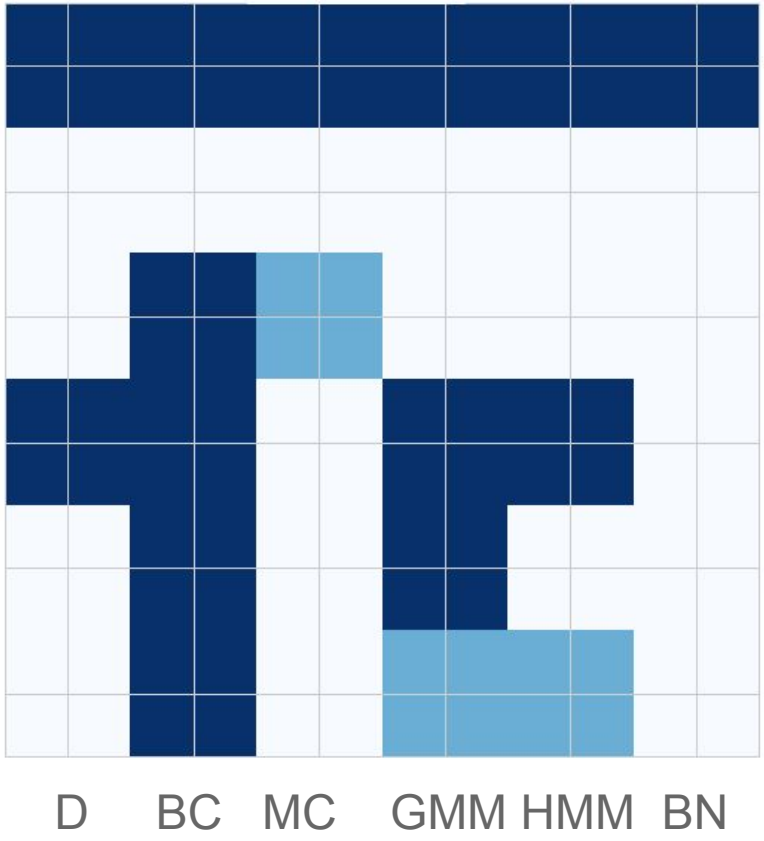
Bayes Classifiers

Markov Chains

General Mixture Models

Hidden Markov Models

Bayesian Networks



Overview: model stacking in pomegranate

Distributions

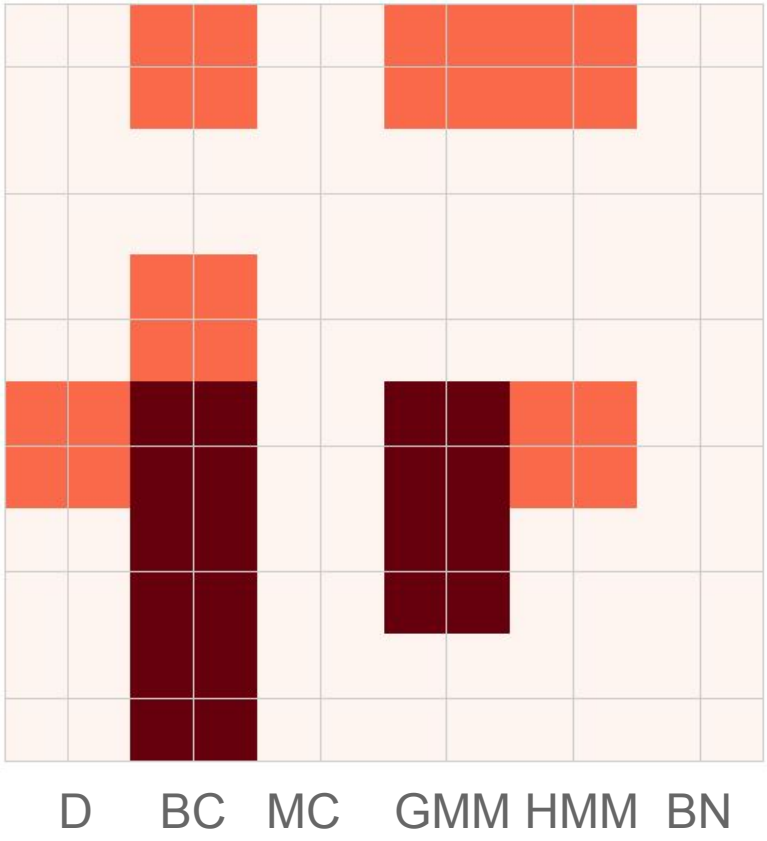
Bayes Classifiers

Markov Chains

General Mixture Models

Hidden Markov Models

Bayesian Networks



The API is common to all models

model.log_probability(X) / model.probability(X)

model.sample()

model.fit(X, weights, inertia)

model.summarize(X, weights)

model.from_summaries(inertia)

model.predict(X)

model.predict_proba(X)

model.predict_log_proba(X)

Model.from_samples(X, weights)

All models have these methods!

All models composed of distributions (like GMM, HMM...) have these methods too!

pomegranate supports many models

Univariate Distributions

1. UniformDistribution
2. BernoulliDistribution
3. NormalDistribution
4. LogNormalDistribution
5. ExponentialDistribution
6. BetaDistribution
7. GammaDistribution
8. DiscreteDistribution
9. PoissonDistribution

Kernel Densities

1. GaussianKernelDensity
2. UniformKernelDensity
3. TriangleKernelDensity

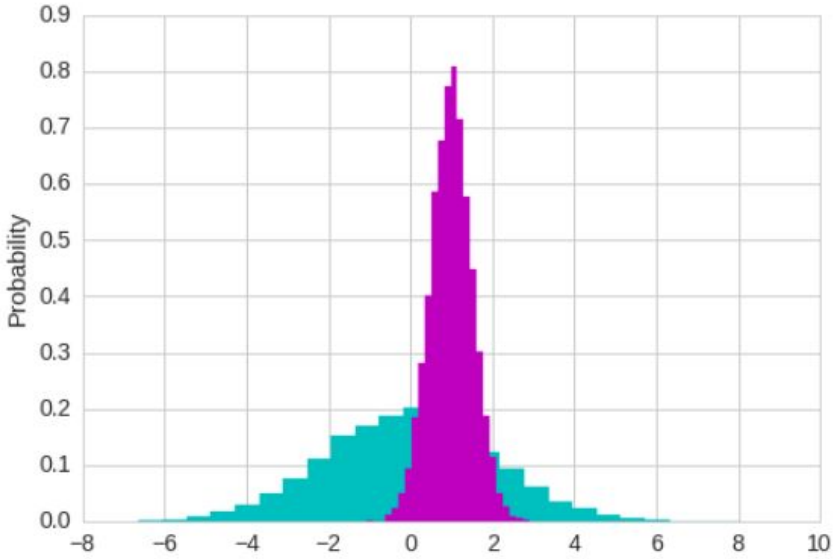
Multivariate Distributions

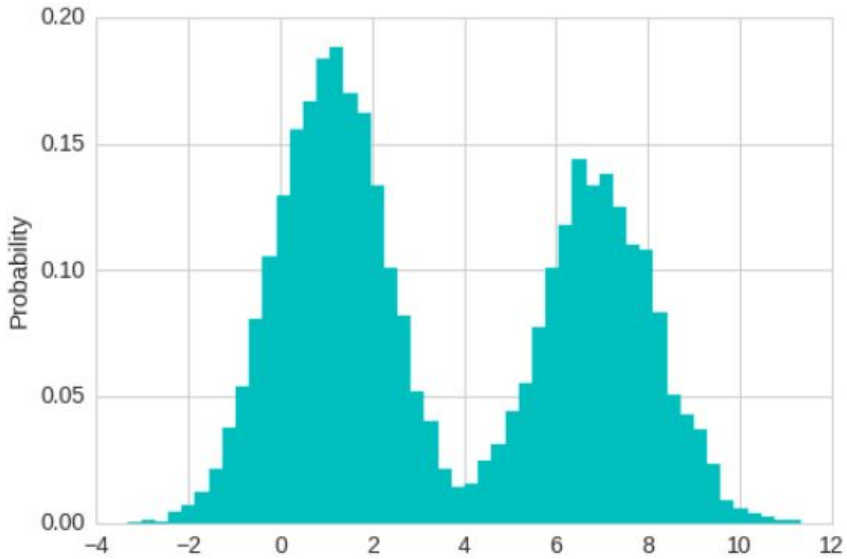
1. IndependentComponentsDistribution
2. MultivariateGaussianDistribution
3. DirichletDistribution
4. ConditionalProbabilityTable
5. JointProbabilityTable

Models can be created from known values

```
mu, sig = 0, 2
a = NormalDistribution(mu, sig)
```

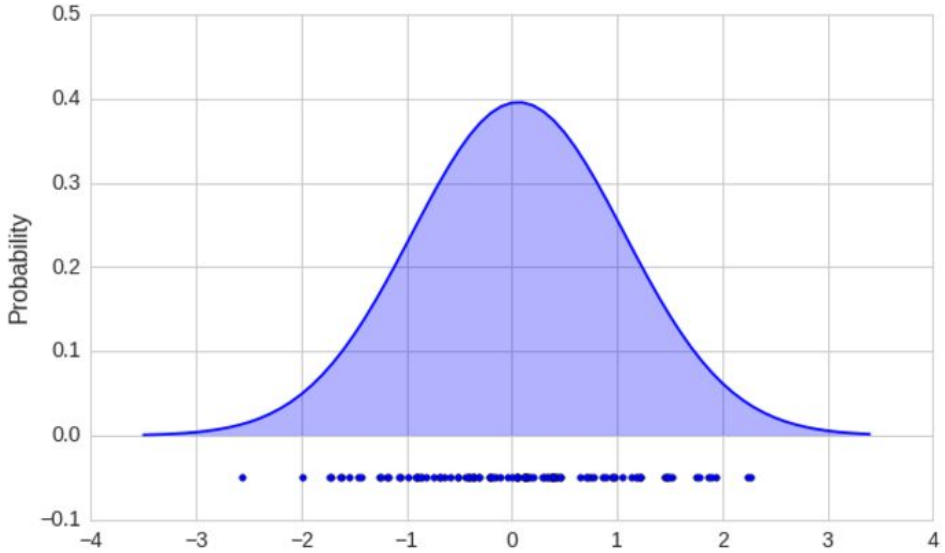
```
X = [0, 1, 1, 2, 1.5, 6, 7, 8, 7]
a = GaussianKernelDensity(X)
```





Models can be learned from data

```
X = numpy.random.normal(0, 1, 100)
a = NormalDistribution.from_samples(X)
```



pomegranate can be faster than numpy

Fitting a Normal Distribution to 1,000 samples

```
data = numpy.random.randn(1000)

print "numpy time:"
%timeit -n 100 data.mean(), data.std()
print
print "pomegranate time:"
%timeit -n 100 NormalDistribution.from_samples(data)
```

```
numpy time:
100 loops, best of 3: 46.6 µs per loop

pomegranate time:
100 loops, best of 3: 22.2 µs per loop
```

pomegranate can be faster than numpy

Fitting Multivariate Gaussian to 10,000,000 samples of 10 dimensions

```
data = numpy.random.randn(10000000, 10)

print "numpy time:"
%timeit -n 10 data.mean(), numpy.cov(data.T)
print
print "pomegranate time:"
%timeit -n 10 MultivariateGaussianDistribution.from_samples(data)
```

```
numpy time:
10 loops, best of 3: 1.02 s per loop

pomegranate time:
10 loops, best of 3: 799 ms per loop
```

pomegranate uses BLAS internally

```
from scipy.linalg.cython_blas cimport dgemm
```

```
dgemm('N', 'T', &d, &d, &n, &alpha, y, &d, X, &d, &beta, pair_sum, &d)
```

pomegranate will soon have GPU support

[WIP] Add GPU support for models #270

Open **jmschrei** wants to merge 1 commit into `master` from `GPU`

Conversation 3 | Commits 1 | Files changed 3

jmschrei commented 12 days ago — Owner

This PR will add GPU support for all models, starting with multivariate gaussian distributions, GMMs, and HMMs, using the package `cupy`. Stay tuned!

pomegranate uses additive summarization

pomegranate reduces data to sufficient statistics for updates and so only has to go datasets once (for all models).

Here is an example of the Normal Distribution sufficient statistics

$$\sum_{i=1}^{n} w_i \quad \sum_{i=1}^{n} w_i x_i \quad \sum_{i=1}^{n} w_i x_i^2 \longrightarrow$$

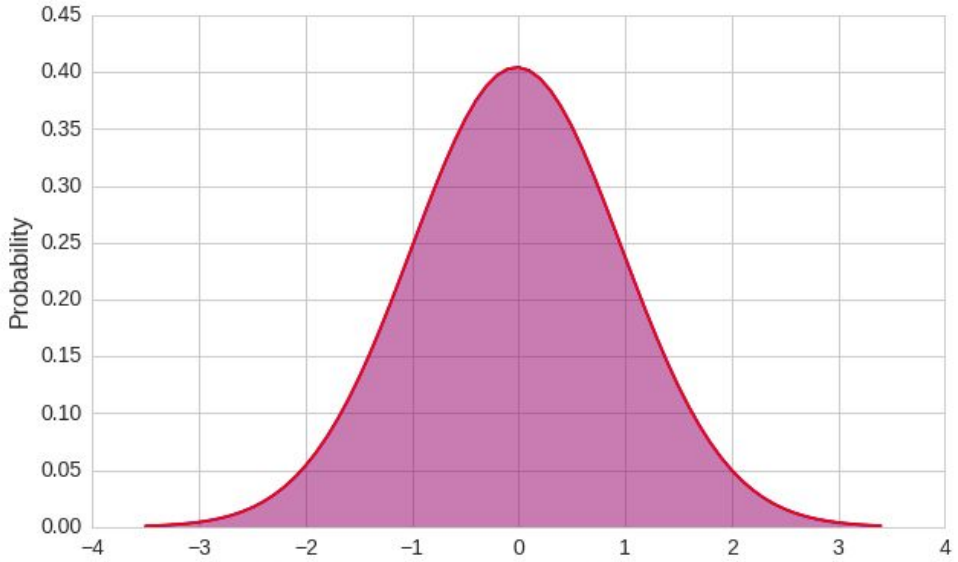
$$\mu = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} w_i x_i^2}{\sum_{i=1}^{n} w_i} - \frac{\left(\sum_{i=1}^{n} w_i x_i\right)^2}{\left(\sum_{i=1}^{n} w_i\right)^2}$$

pomegranate supports out-of-core learning

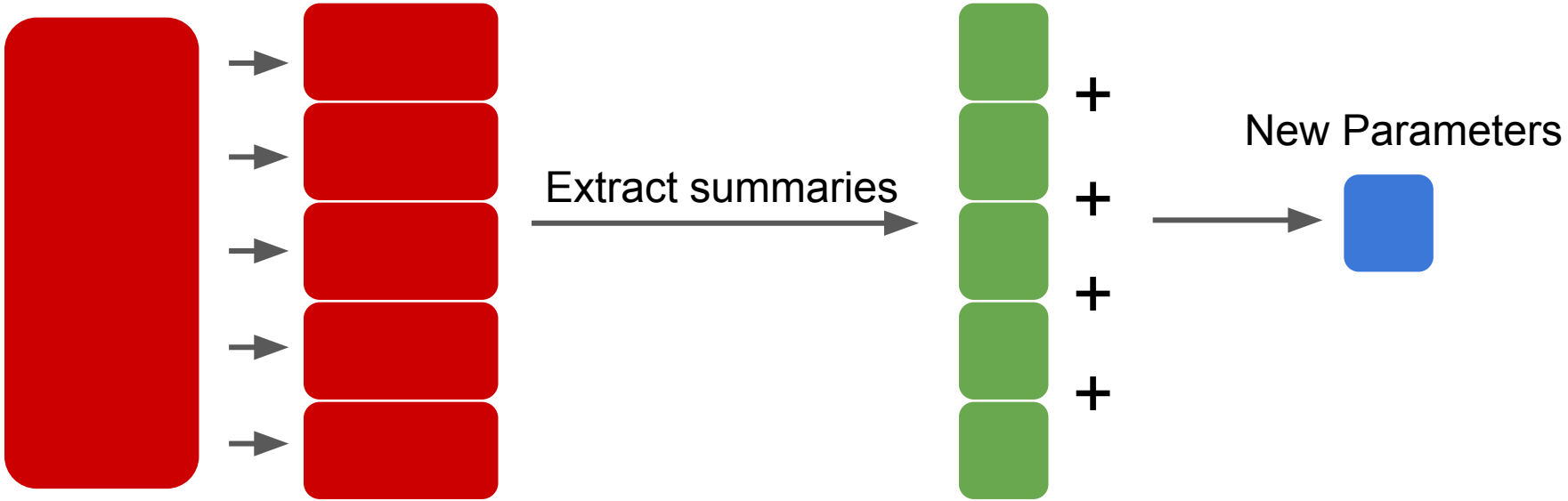
Batches from a dataset can be reduced to additive summary statistics, enabling exact updates from data that can't fit in memory.

```
a.fit(data)
b.summarize(data[:1000])
b.summarize(data[1000:2000])
b.summarize(data[2000:3000])
b.summarize(data[3000:4000])
b.summarize(data[4000:])
b.from_summaries()
```



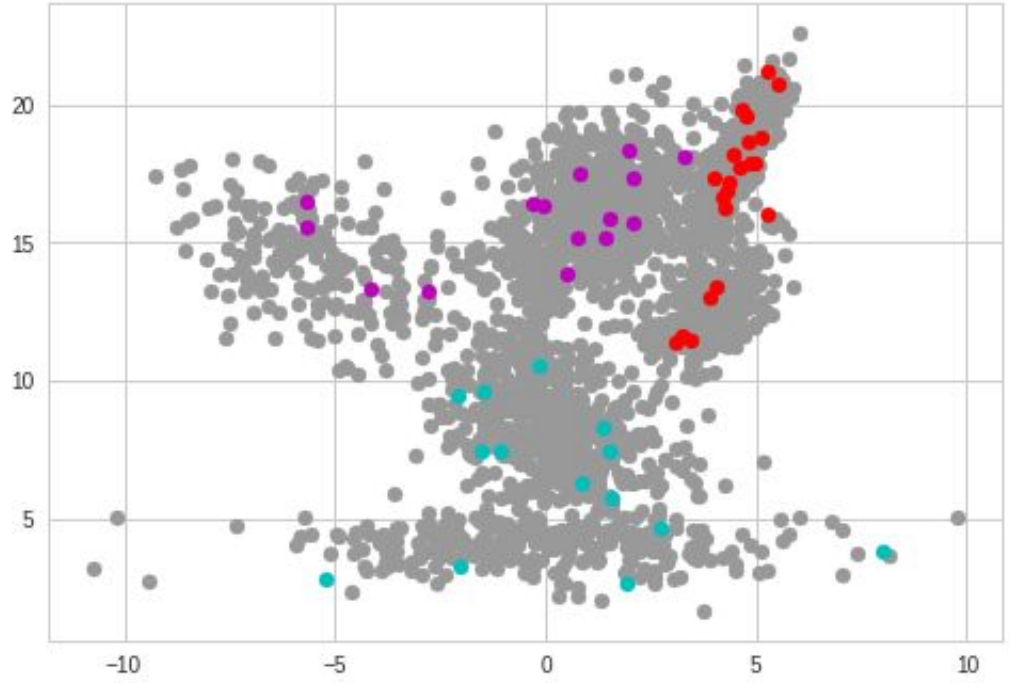

```
Fit Mean:       -0.0174820965846, Fit STD:       0.986767322871
Summarize Mean: -0.0174820965846, Summarize STD: 0.986767322871
```

Parallelization exploits additive summaries



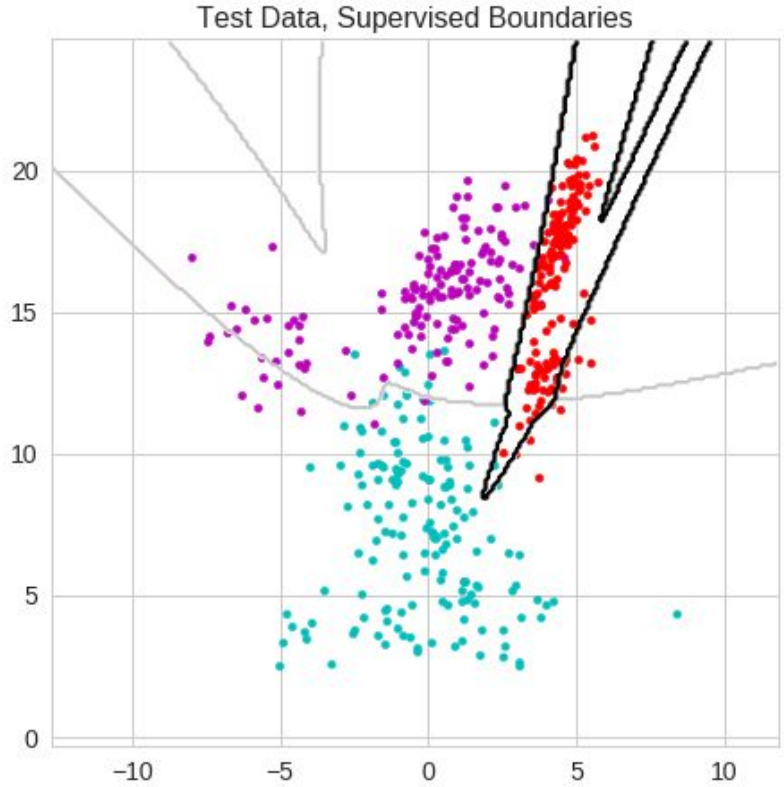
pomegranate supports semisupervised learning

Summary statistics from supervised models can be added to summary statistics from unsupervised models to train a single model on a mixture of labeled and unlabeled data.

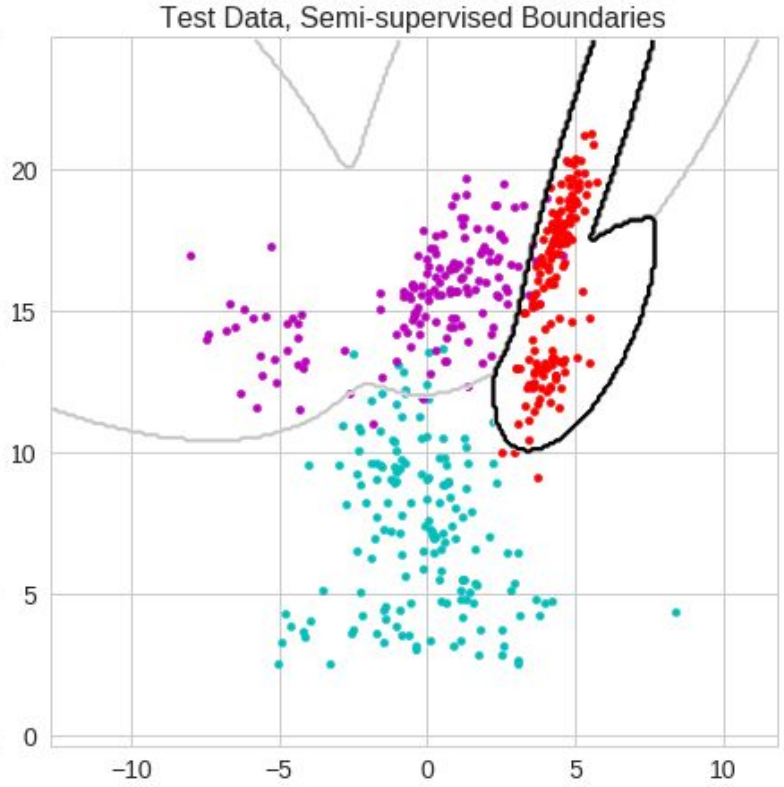


pomegranate supports semisupervised learning

Supervised Accuracy: 0.93



Semisupervised Accuracy: 0.96



pomegranate can be faster than scipy

```
mu, cov = numpy.random.randn(2000), numpy.eye(2000)
d = MultivariateGaussianDistribution(mu, cov)
X = numpy.random.randn(2000, 2000)
print "scipy time: ",
%timeit multivariate_normal.logpdf(X, mu, cov)
print "pomegranate time: ",
%timeit MultivariateGaussianDistribution(mu, cov).log_probability(X)
print "pomegranate time (w/ precreated object): ",
%timeit d.log_probability(X)
```

```
scipy time: 1 loop, best of 3: 1.67 s per loop
 pomegranate time: 1 loop, best of 3: 801 ms per loop
 pomegranate time (w/ precreated object): 1 loop, best of 3: 216 ms per loop
```

pomegranate uses aggressive caching

$$P(X|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$logP(X|\mu,\sigma) = -\log\left(\sqrt{2\pi}\sigma\right) - \frac{(x-\mu)^2}{2\sigma^2}$$

$$logP(X|\mu,\sigma) = \alpha - \frac{(x-\mu)^2}{\beta}$$


GOSSIP GIRL

Example ‘blast’ from Gossip Girl

Spotted: Lonely Boy. Can't believe the love of his life has returned. If only she knew who he was. But everyone knows Serena. And everyone is talking. Wonder what Blair Waldorf thinks. Sure, they're BFF's, but we always thought Blair's boyfriend Nate had a thing for Serena.

Example ‘blast’ from Gossip Girl

Why'd she leave? Why'd she return? Send me all the deets. And who am I? That's the secret I'll never tell. The only one. —XOXO. Gossip Girl.

How do we encode these ‘blasts’?

Better lock it down with Nate, B. Clock's ticking.

+1 Nate
-1 Blair

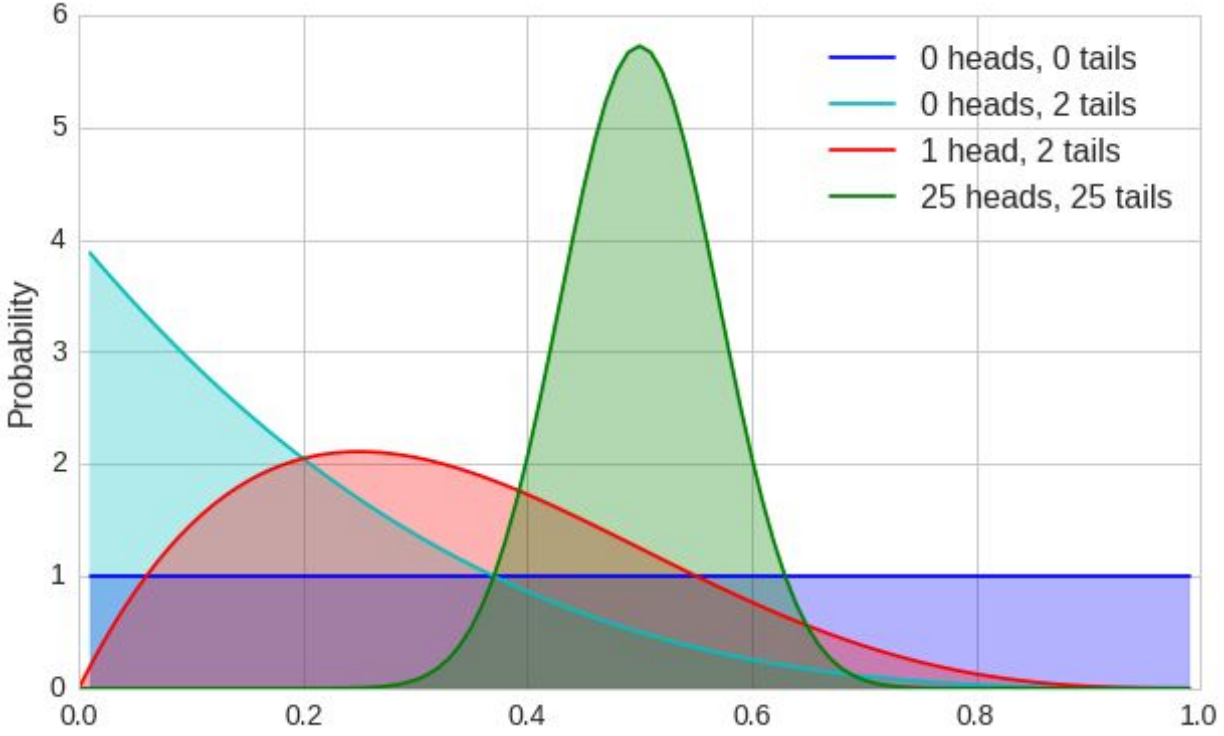
How do we encode these ‘blasts’?

This just in: S and B committing a crime of fashion. Who doesn't love a five-finger discount. Especially if it's the middle one.

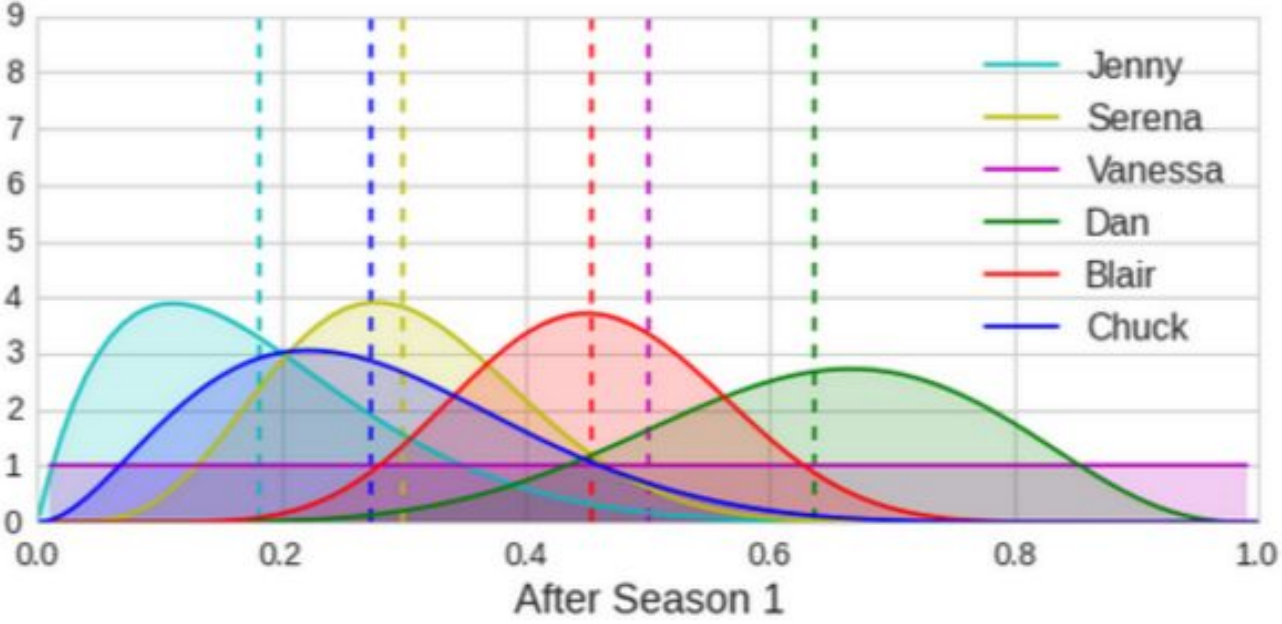
-1 Blair
-1 Serena

Simple summations don’t work well

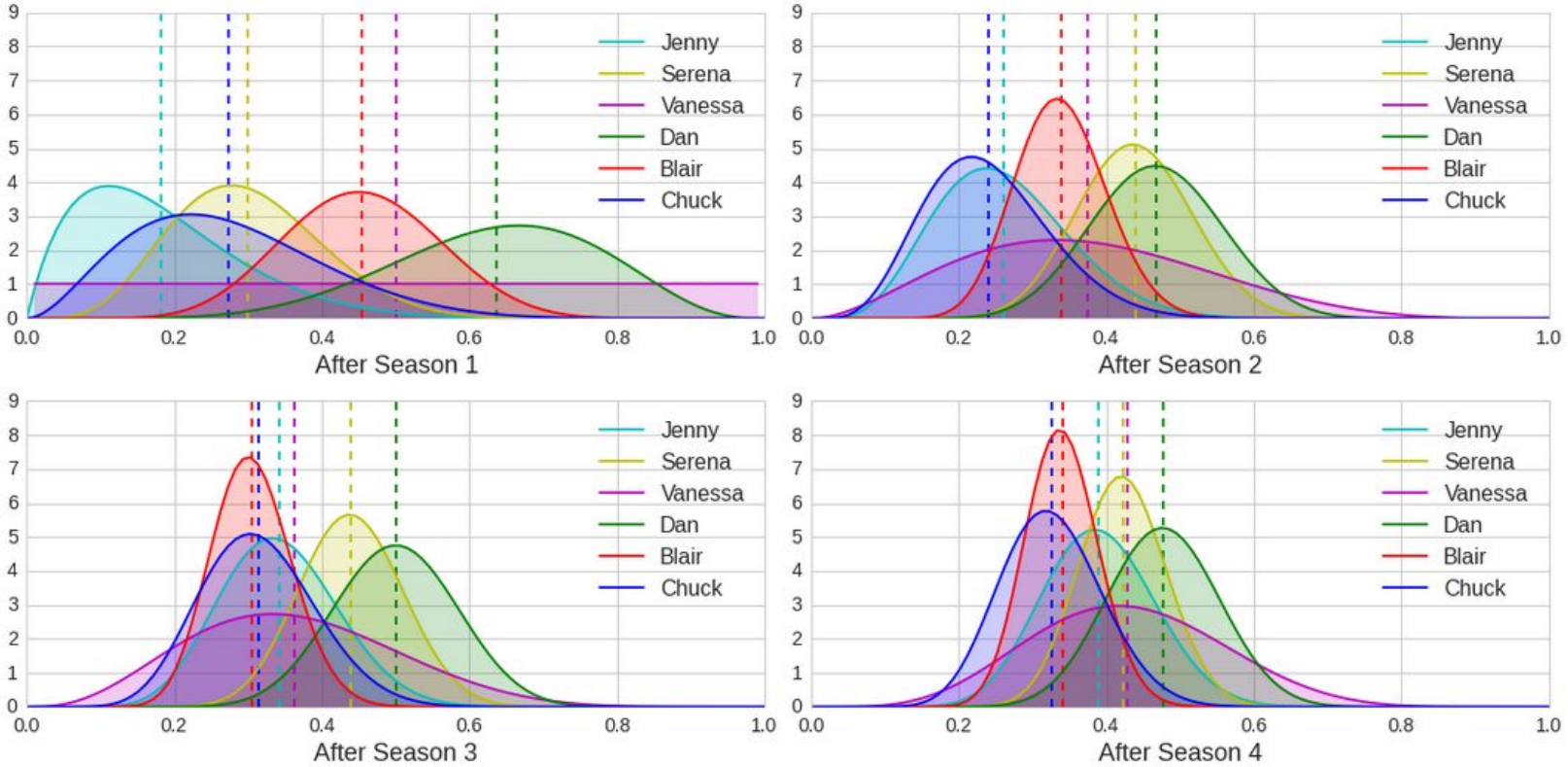
Beta distributions can model uncertainty



Beta distributions can model uncertainty



Beta distributions can model uncertainty



Overview: this talk

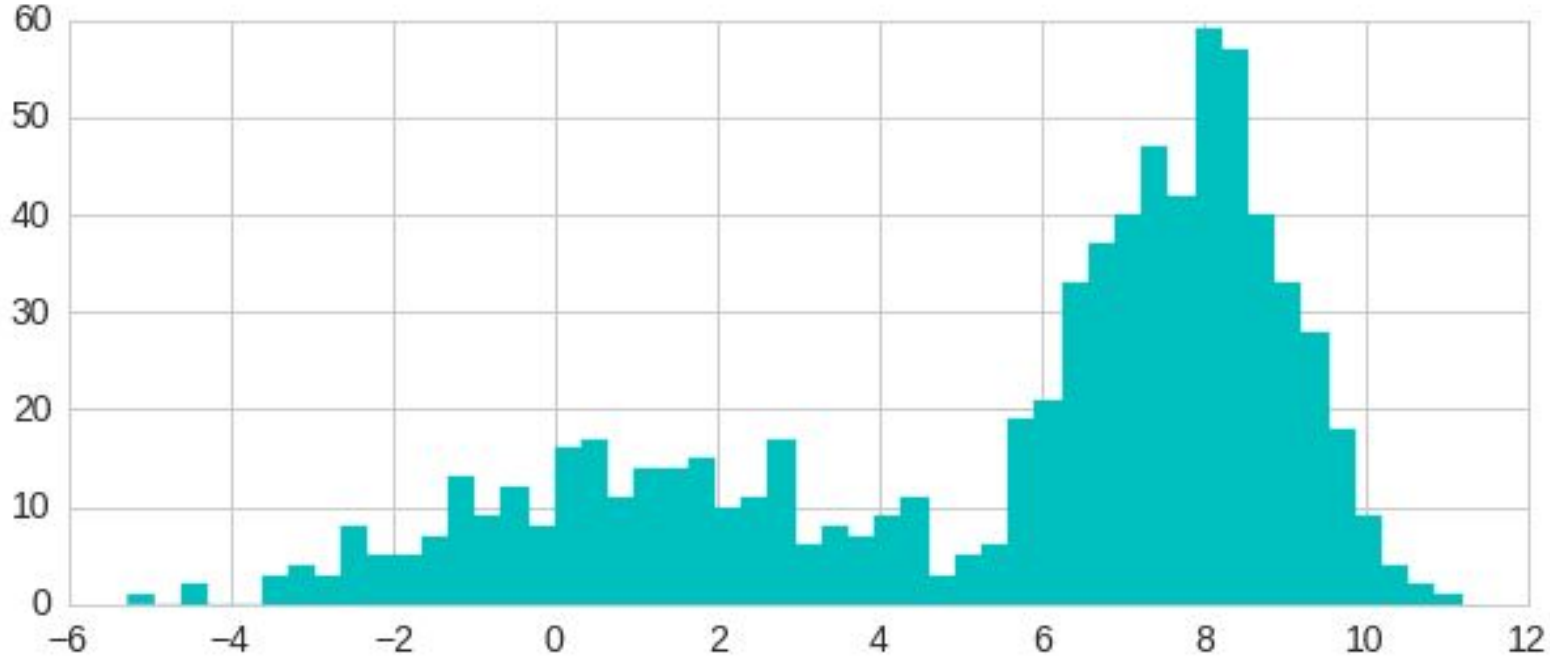
Overview

Major Models/Model Stacks

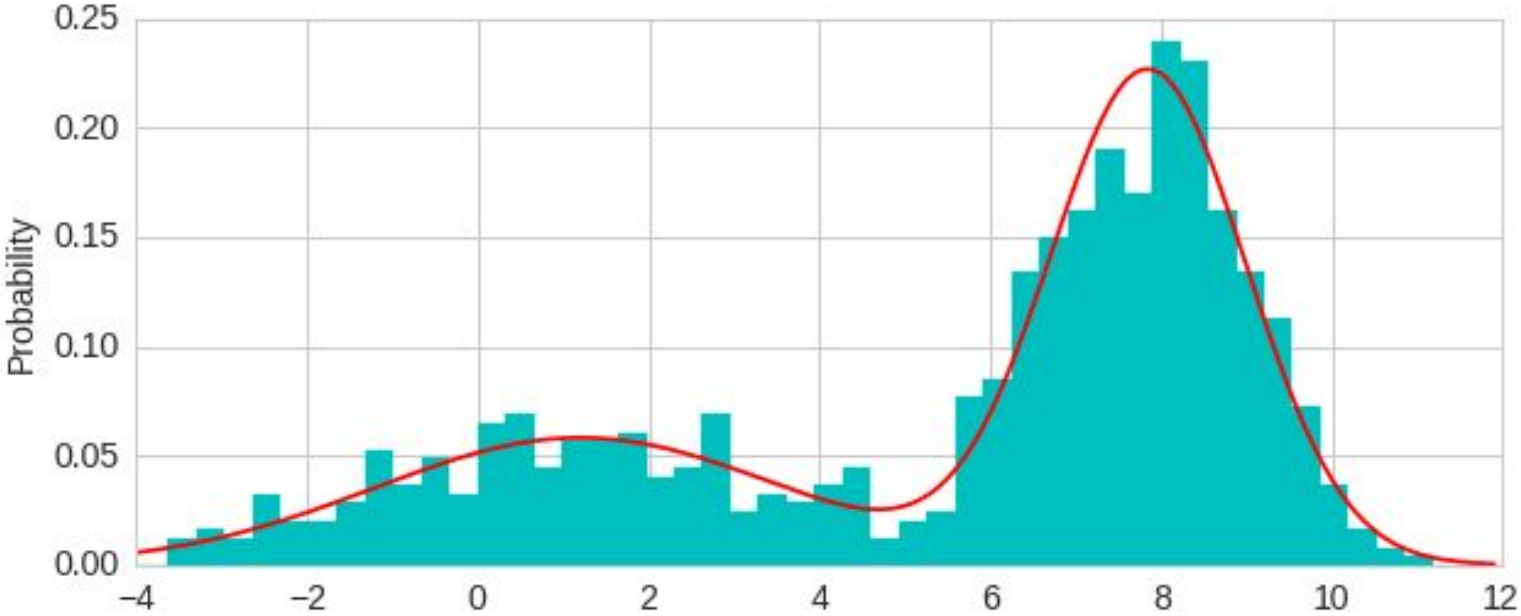
1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

GMMs can model complex distributions

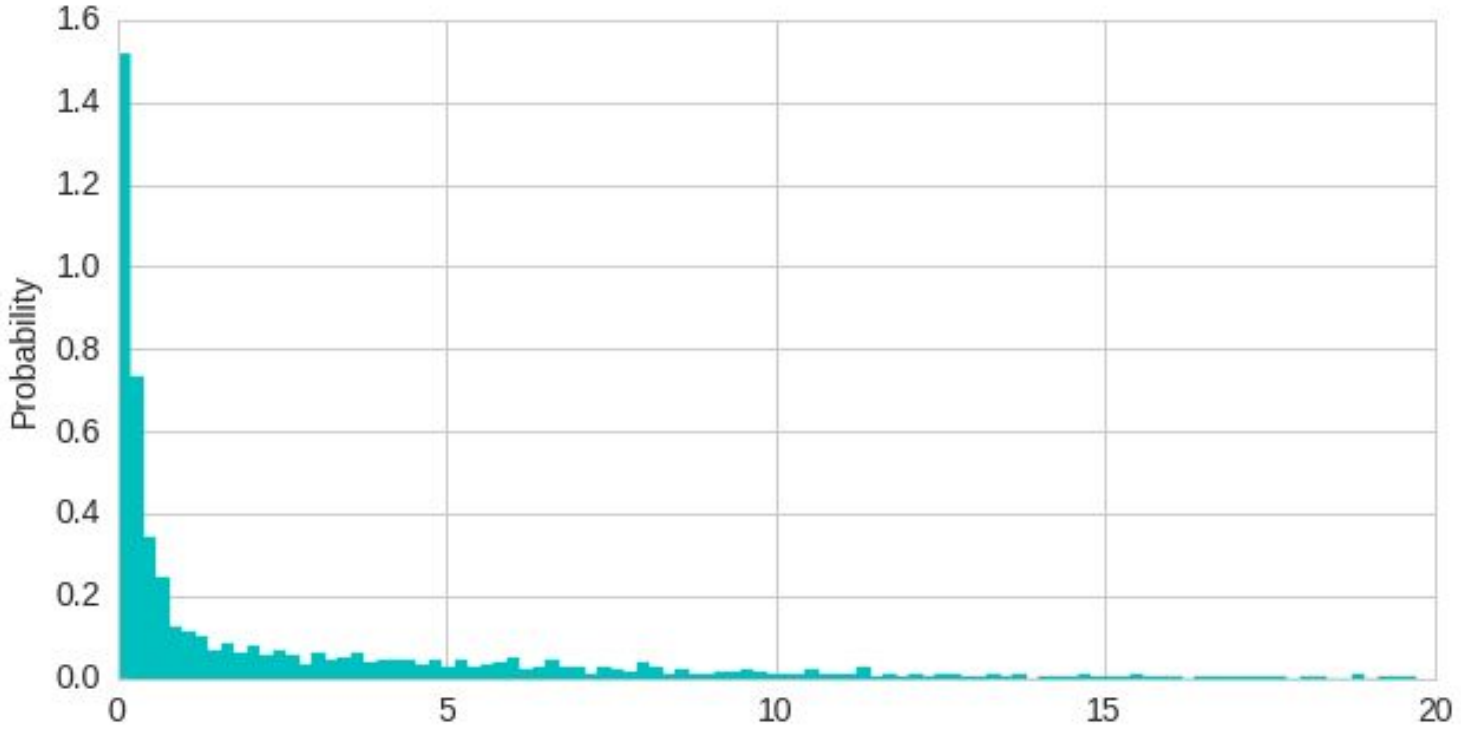


GMMs can model complex distributions

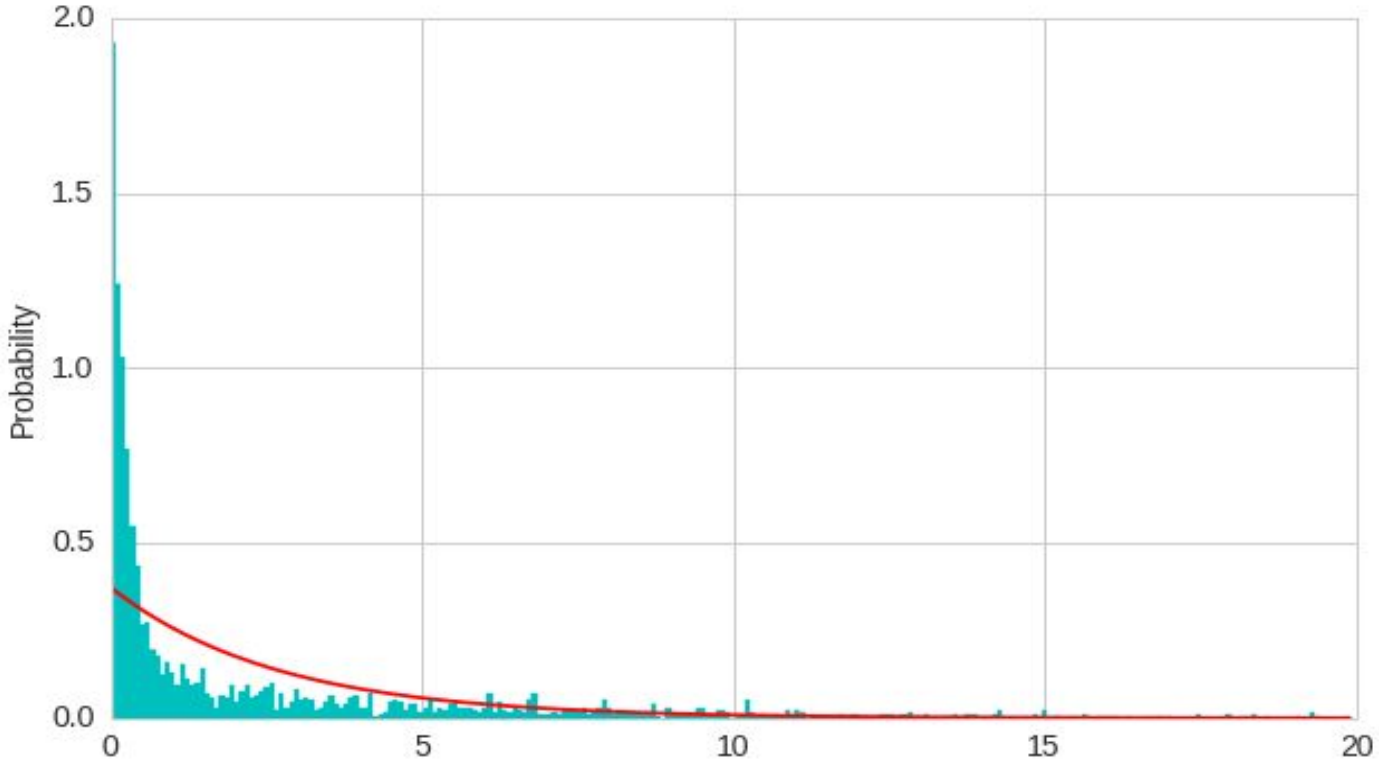


model = GeneralMixtureModel.from_samples(NormalDistribution, 2, X)

GMMs can model complex distributions

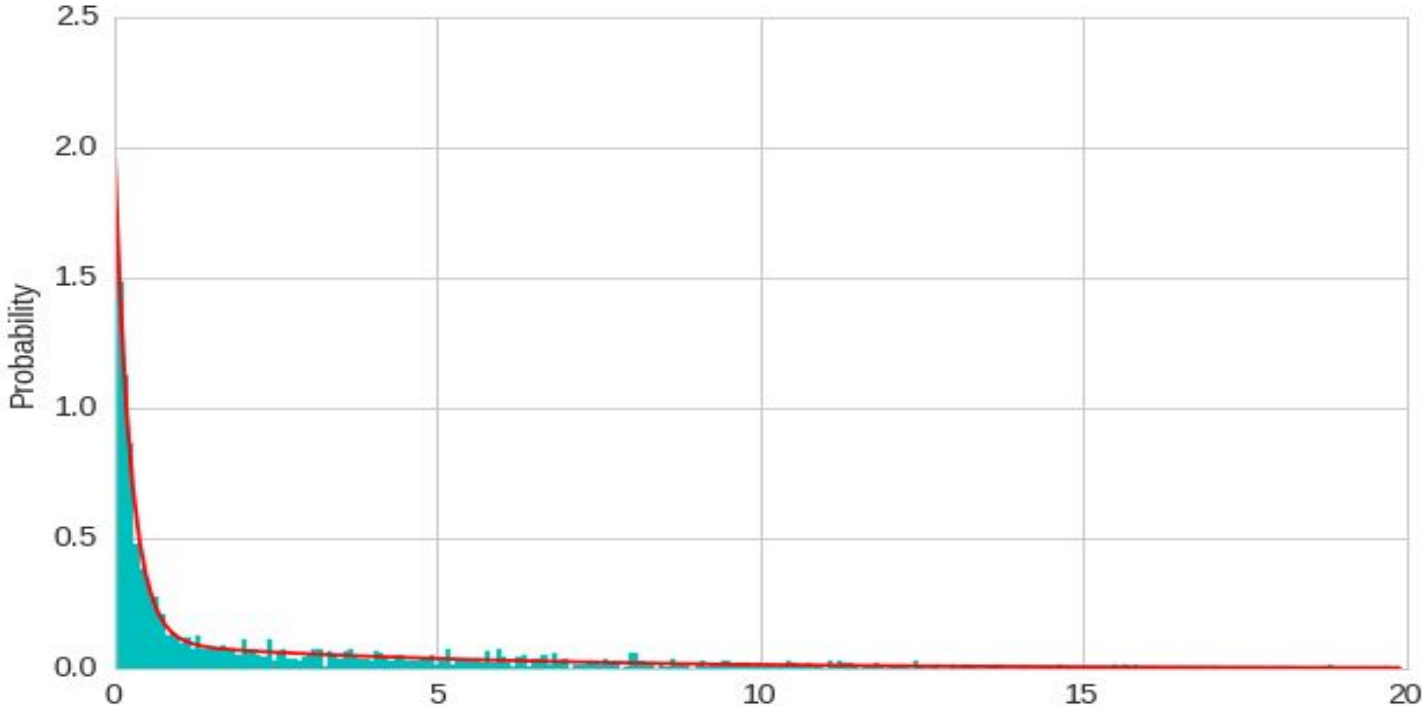


An exponential distribution is not right



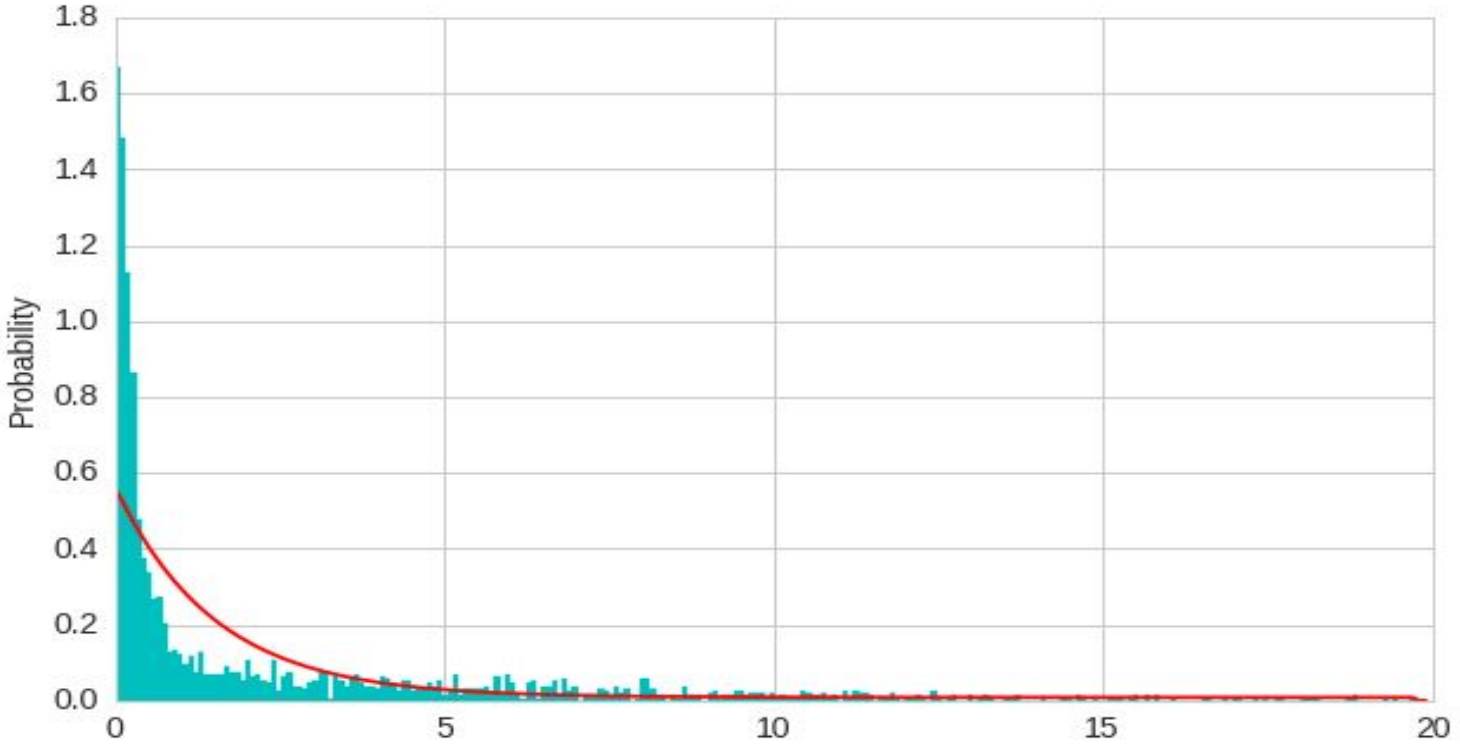
model = ExponentialDistribution.from_samples(X)

A mixture of exponentials is better



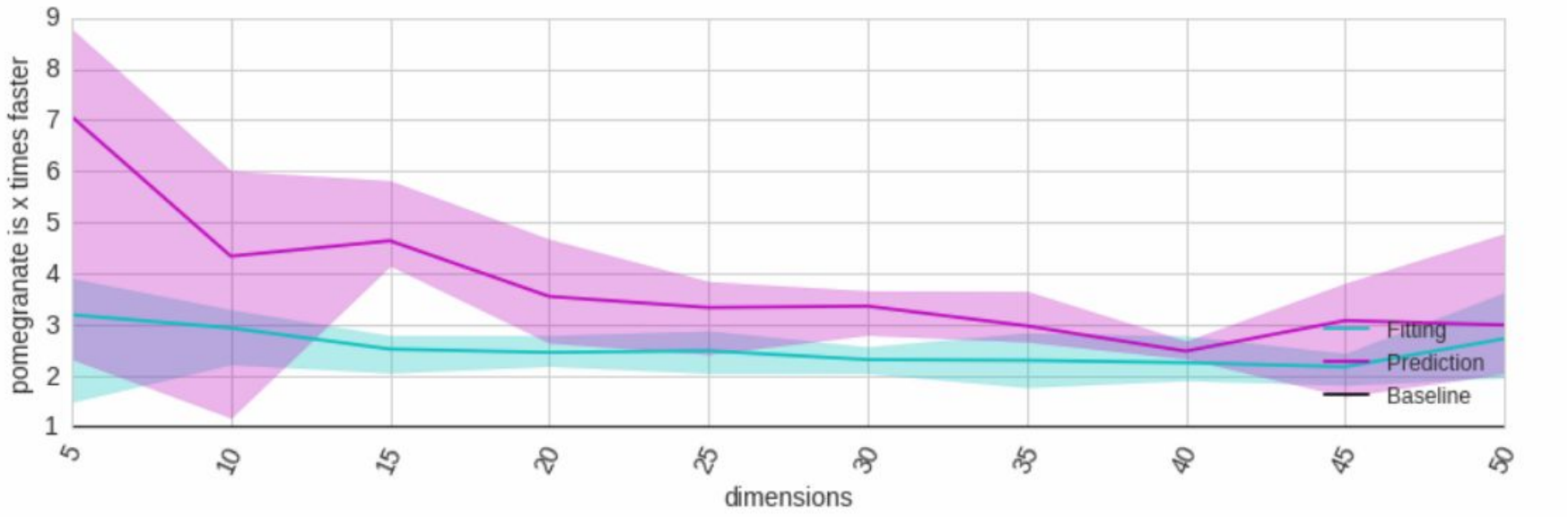
model = GeneralMixtureModel.from_samples(ExponentialDistribution, 2, X)

Heterogeneous mixtures natively supported



model = GeneralMixtureModel.from_samples([ExponentialDistribution, UniformDistribution], 2, X)

GMMs faster than sklearn



Overview: this talk

Overview

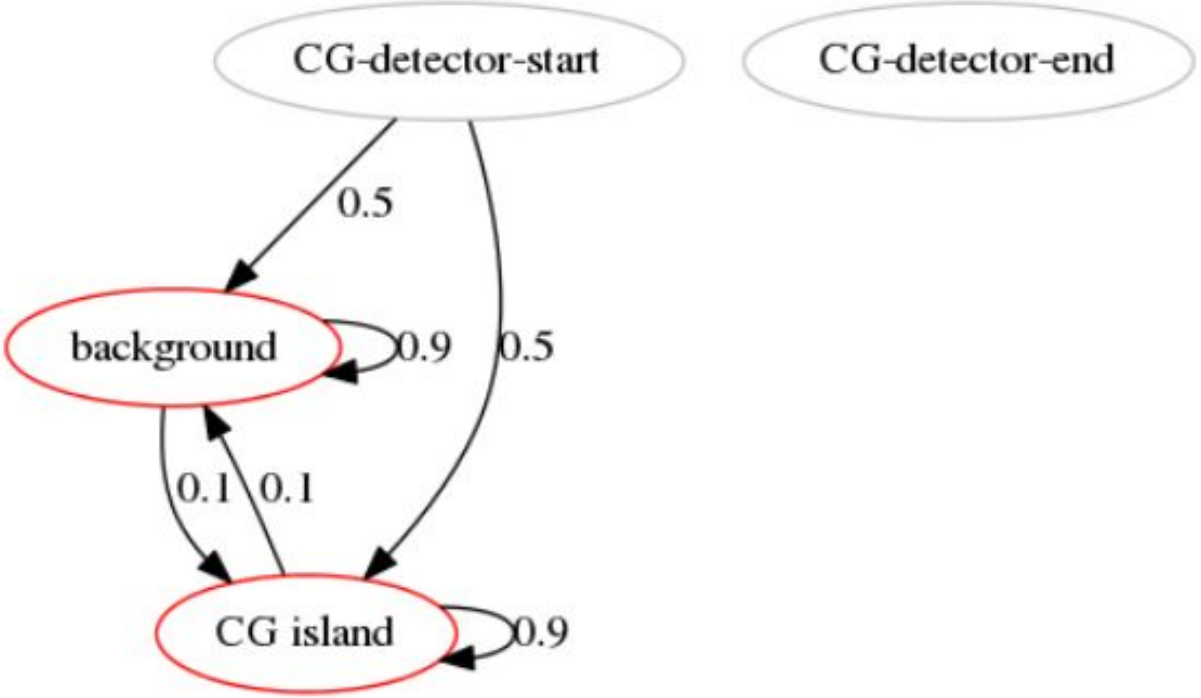
Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

CG enrichment detection HMM

GACTACGACTCGCGCTCGCACGTCGCTCGACATCATCGACA



CG enrichment detection HMM

GACTACGACTCGCGCTCGCACGTCGCTCGACATCATCGACA

```
d1 = DiscreteDistribution({'A': 0.25, 'C': 0.25, 'G': 0.25, 'T': 0.25})
d2 = DiscreteDistribution({'A': 0.10, 'C': 0.40, 'G': 0.40, 'T': 0.10})

s1 = State(d1, name="background")
s2 = State(d2, name="CG island")

hmm = HiddenMarkovModel("CG-detector")
hmm.add_states(s1, s2)
hmm.add_transition(hmm.start, s1, 0.5)
hmm.add_transition(hmm.start, s2, 0.5)
hmm.add_transition(s1, s1, 0.9)
hmm.add_transition(s1, s2, 0.1)
hmm.add_transition(s2, s1, 0.1)
hmm.add_transition(s2, s2, 0.9)
hmm.bake()
```

pomegranate HMMs are feature rich

Feature	pomegranate	hmmlearn
Graph Structure		
Silent States	✓	
Optional Explicit End State	✓	
Sparse Implementation	✓	
Arbitrary Emissions Allowed on States	✓	
Discrete/Gaussian/GMM Emissions	✓	✓
Large Library of Other Emissions	✓	
Build Model from Matrices	✓	✓
Build Model Node-by-Node	✓	
Serialize to JSON	✓	
Serialize using Pickle/Joblib	✓	✓

Algorithms		
Priors		✓
Sampling	✓	✓
Log Probability Scoring	✓	✓
Forward-Backward Emissions	✓	✓
Forward-Backward Transitions	✓	
Viterbi Decoding	✓	✓
MAP Decoding	✓	✓
Baum-Welch Training	✓	✓
Viterbi Training	✓	
Labeled Training	✓	
Tied Emissions	✓	
Tied Transitions	✓	
Emission Inertia	✓	
Transition Inertia	✓	
Emission Freezing	✓	✓
Transition Freezing	✓	✓
Multi-threaded Training	✓	

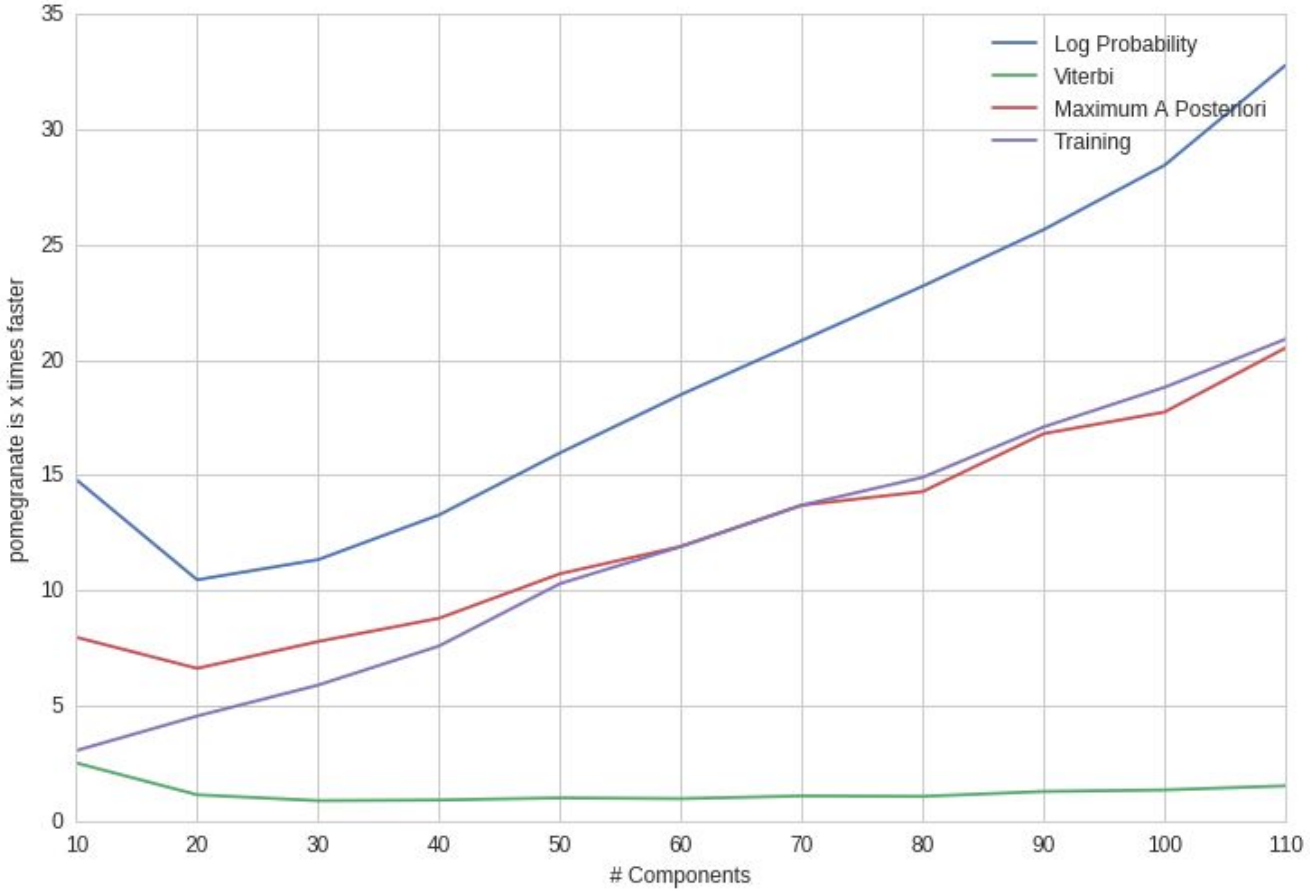
GMM-HMM easy to define

```
d1 = GeneralMixtureModel([NormalDistribution(5, 2), NormalDistribution(5, 4)])
d2 = GeneralMixtureModel([NormalDistribution(15, 1), NormalDistribution(15, 5)])

s1 = State(d1, name="GMM1")
s2 = State(d2, name="GMM2")

model = HiddenMarkovModel()
model.add_states(s1, s2)
model.add_transition(model.start, s1, 0.75)
model.add_transition(model.start, s2, 0.25)
model.add_transition(s1, s1, 0.85)
model.add_transition(s1, s2, 0.15)
model.add_transition(s2, s2, 0.90)
model.add_transition(s2, s1, 0.10)
model.bake()
```

HMMs are faster than hmmlearn



Overview: this talk

Overview

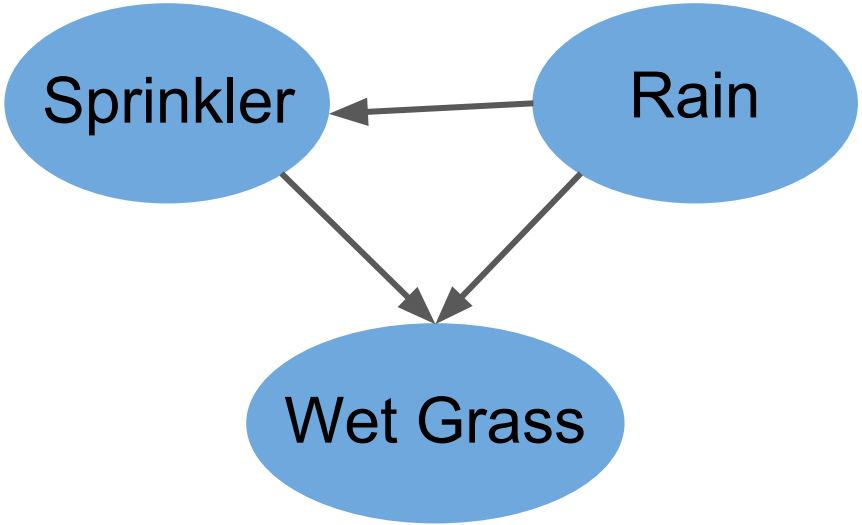
Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

Bayesian networks

Bayesian networks are powerful inference tools which define a dependency structure between variables.

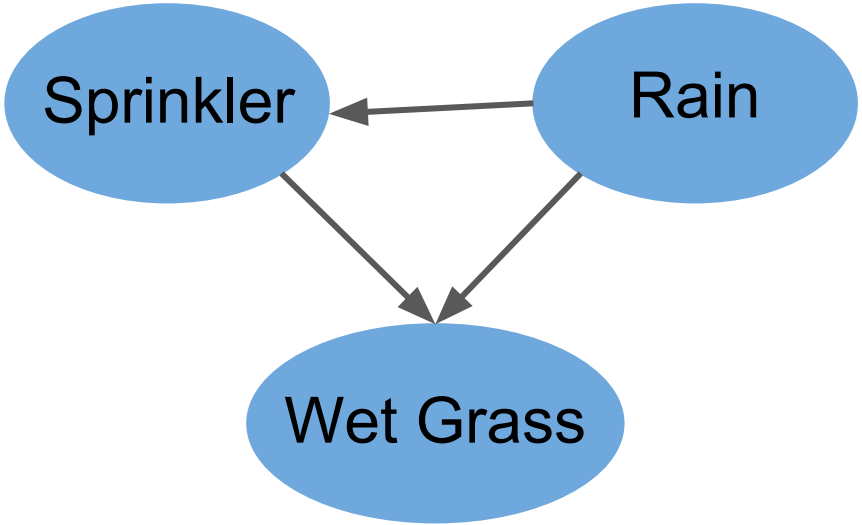


Bayesian networks

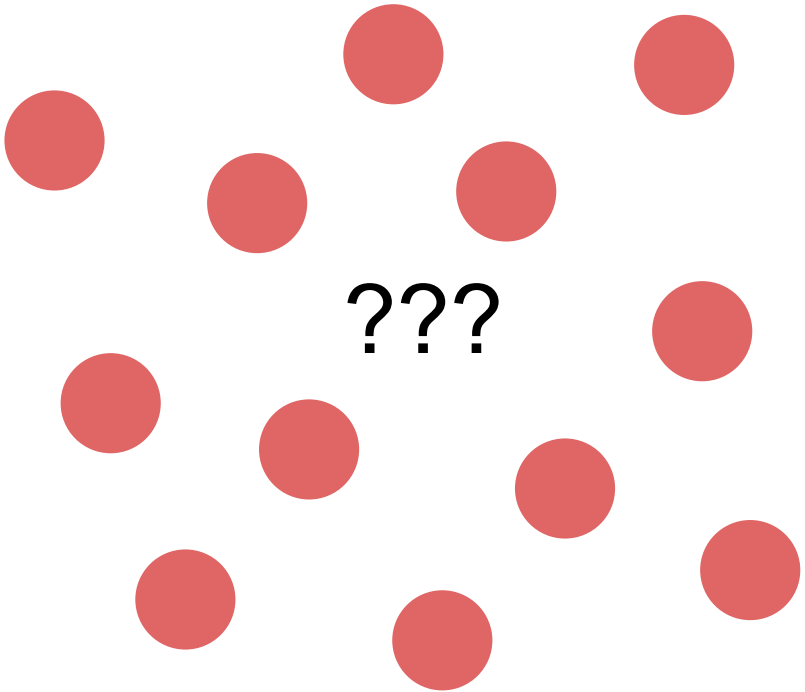
Two main difficult tasks:

(1) Inference given incomplete information

(2) Learning the dependency structure from data



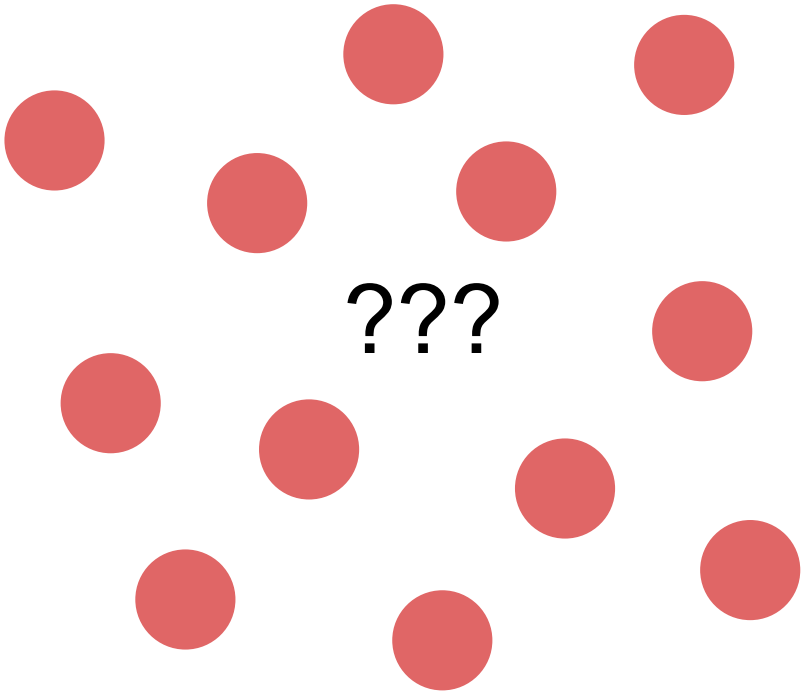
Bayesian network structure learning



Three primary ways:

- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics

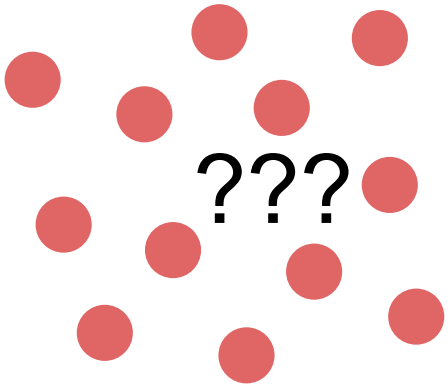
Bayesian network structure learning

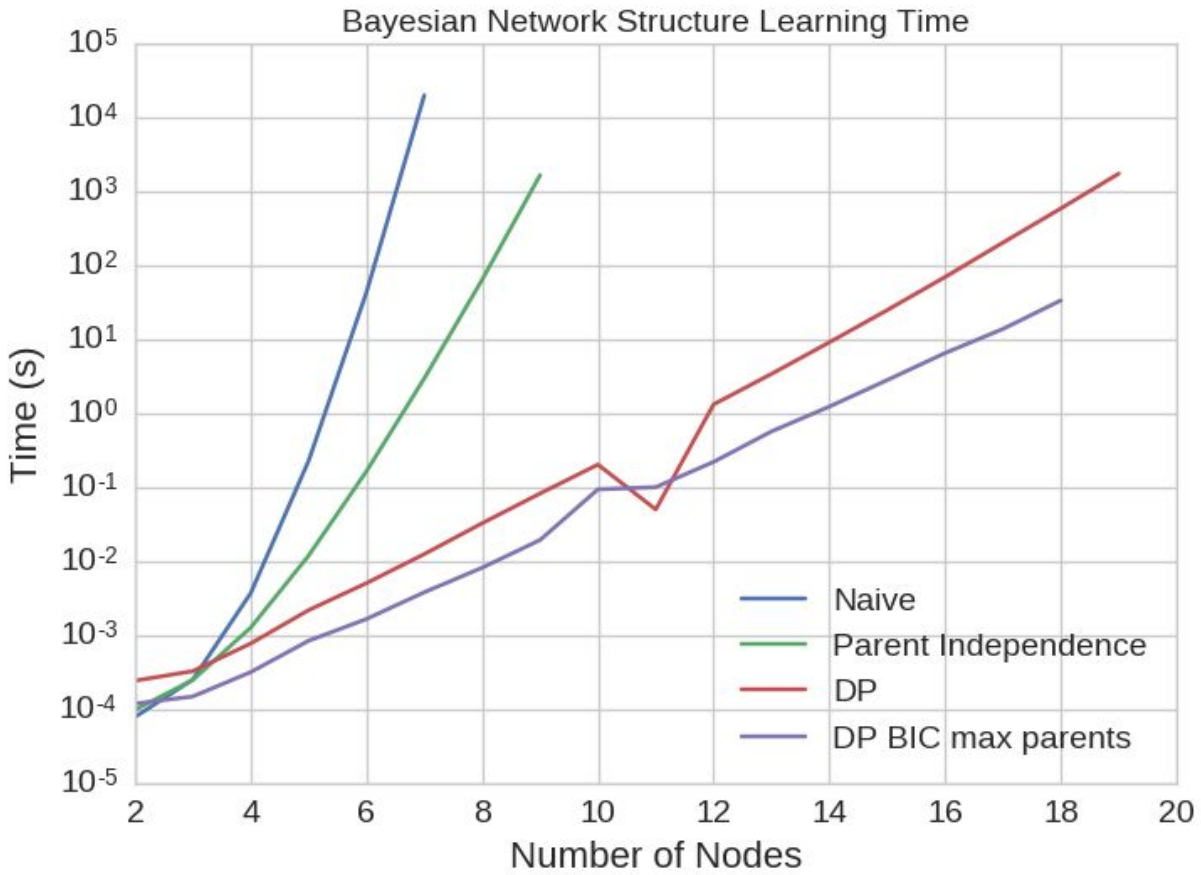


pomegranate supports:

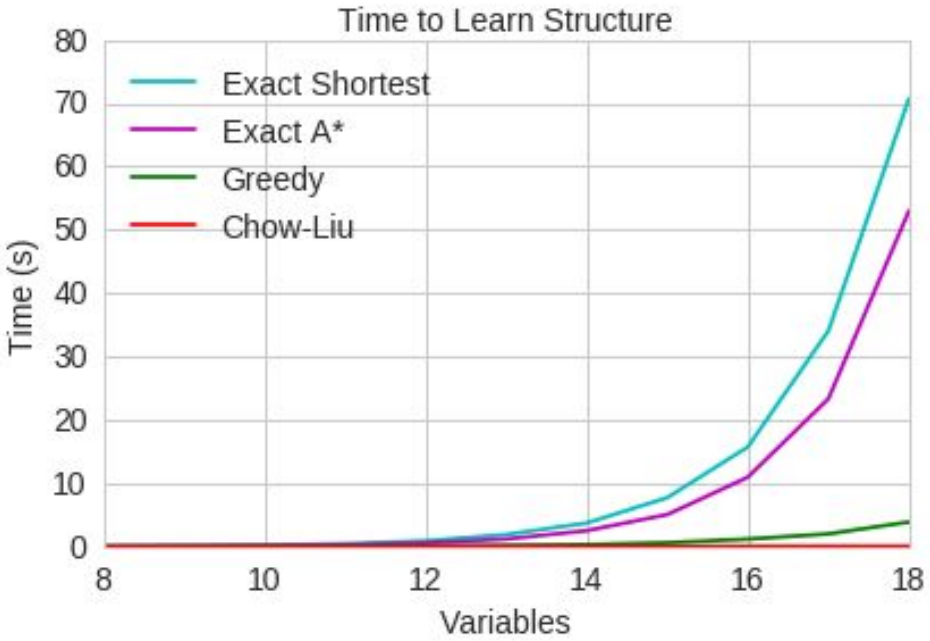
- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics

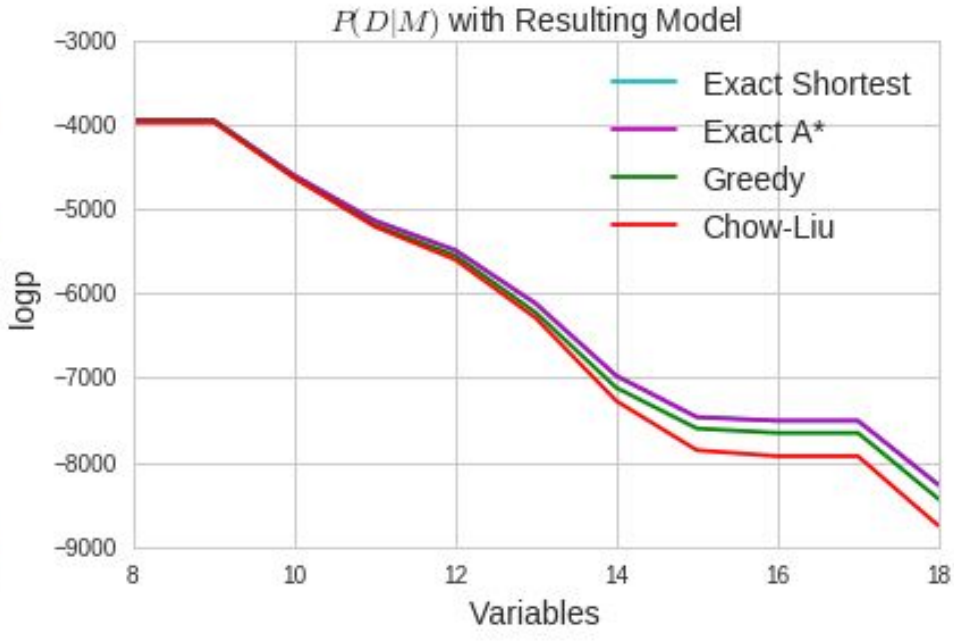
Exact structure learning is intractable



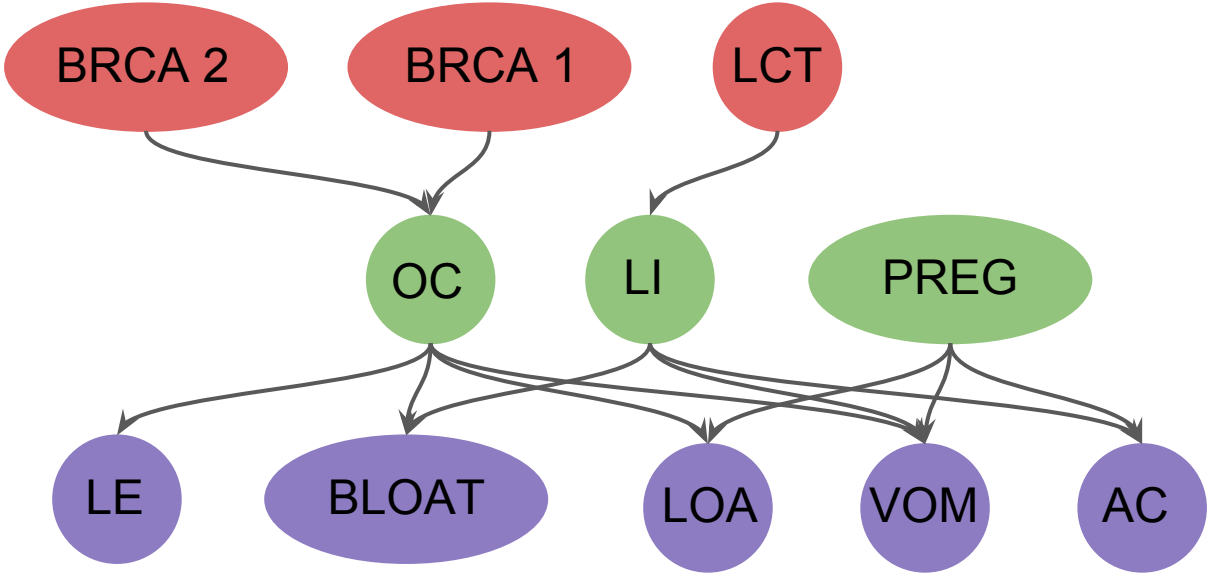


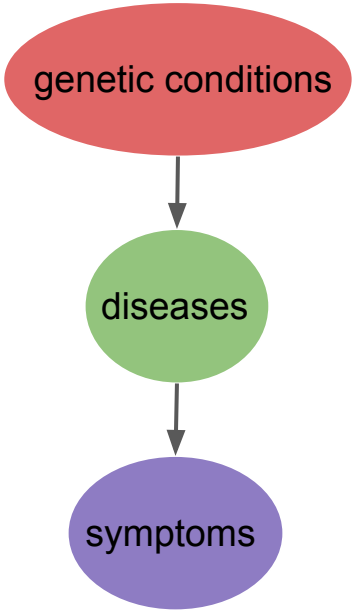
pomegranate supports four algorithms



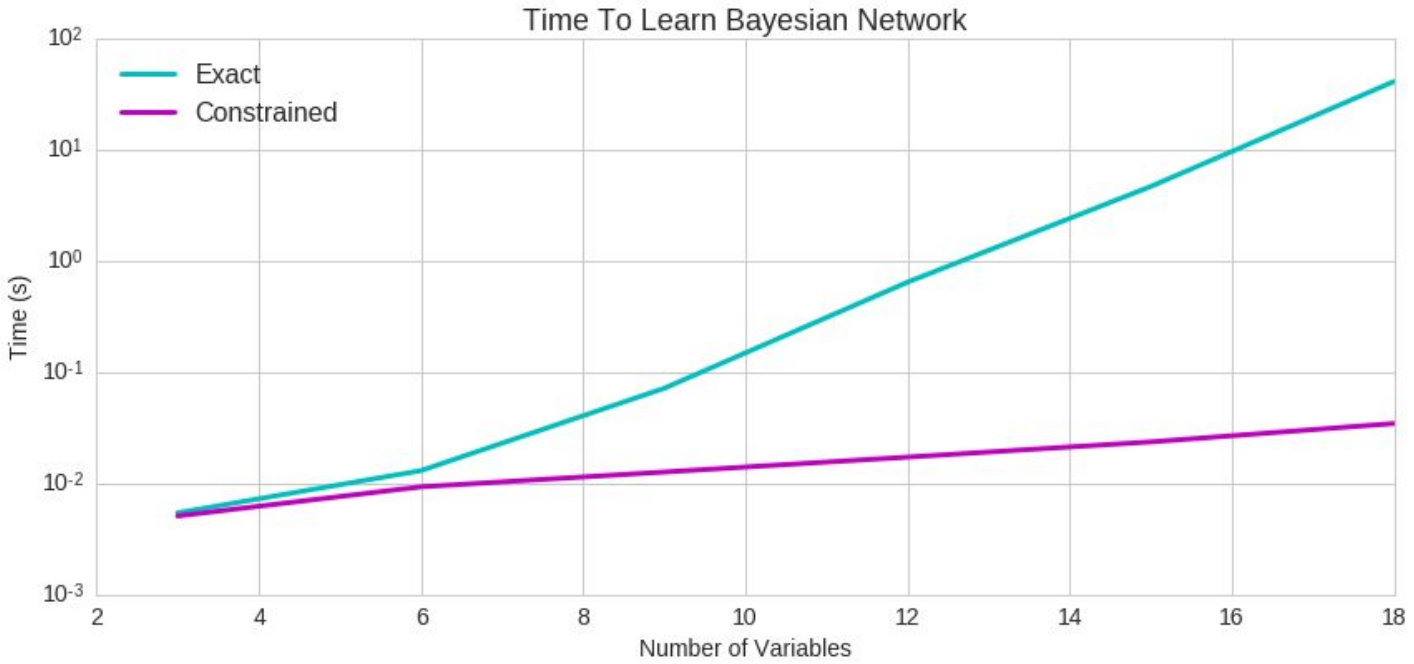


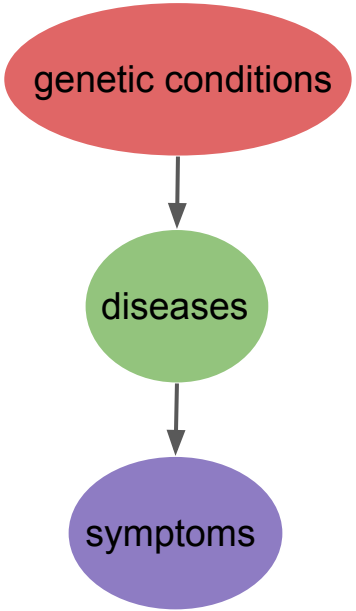
Constraint graphs merge data + knowledge



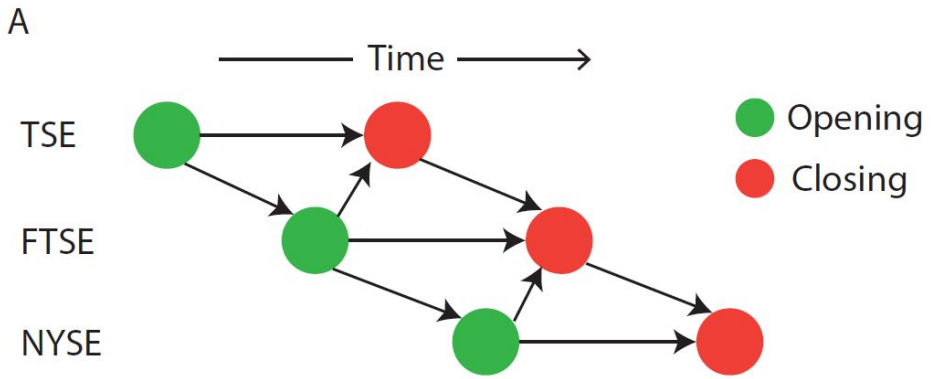


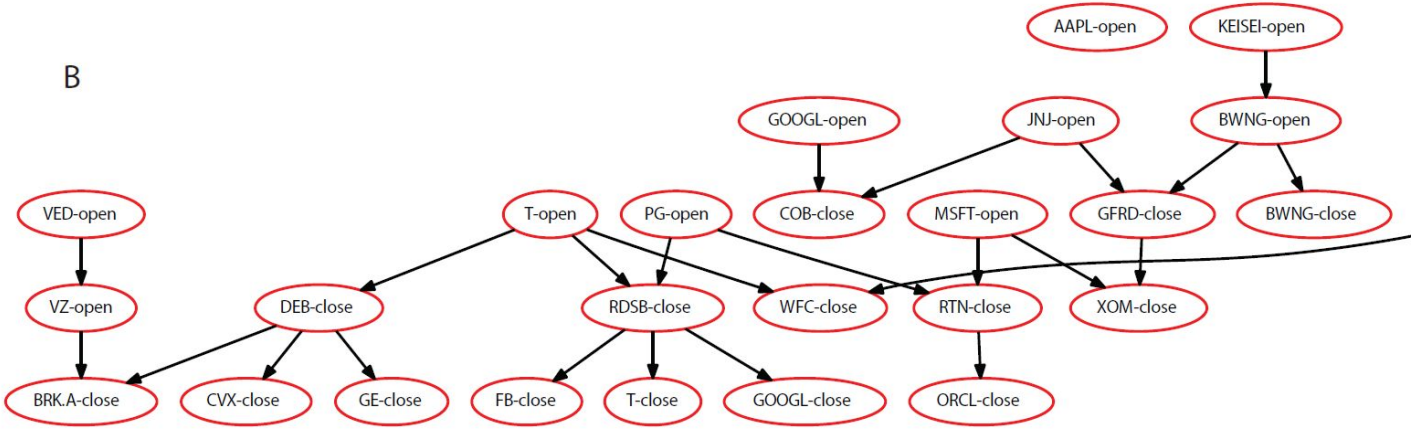
Constraint graphs merge data + knowledge





Modeling the global stock market





Constraint graph published in PeerJ CS

Finding the optimal Bayesian network given a constraint graph

Jacob M. Schreiber[1] and William S. Noble[2]

[1] Department of Computer Science, University of Washington, Seattle, WA, United States of America
[2] Department of Genome Science, University of Washington, Seattle, WA, United States of America

ABSTRACT

Despite recent algorithmic improvements, learning the optimal structure of a Bayesian network from data is typically infeasible past a few dozen variables. Fortunately, domain knowledge can frequently be exploited to achieve dramatic computational savings, and in many cases domain knowledge can even make structure learning tractable. Several methods have previously been described for representing this type of structural prior

Overview: this talk

Overview

Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

Bayes classifiers rely on Bayes' rule

$$P(M|D) = \frac{P(D|M)P(M)}{\sum_{M} P(D|M)P(M)}$$

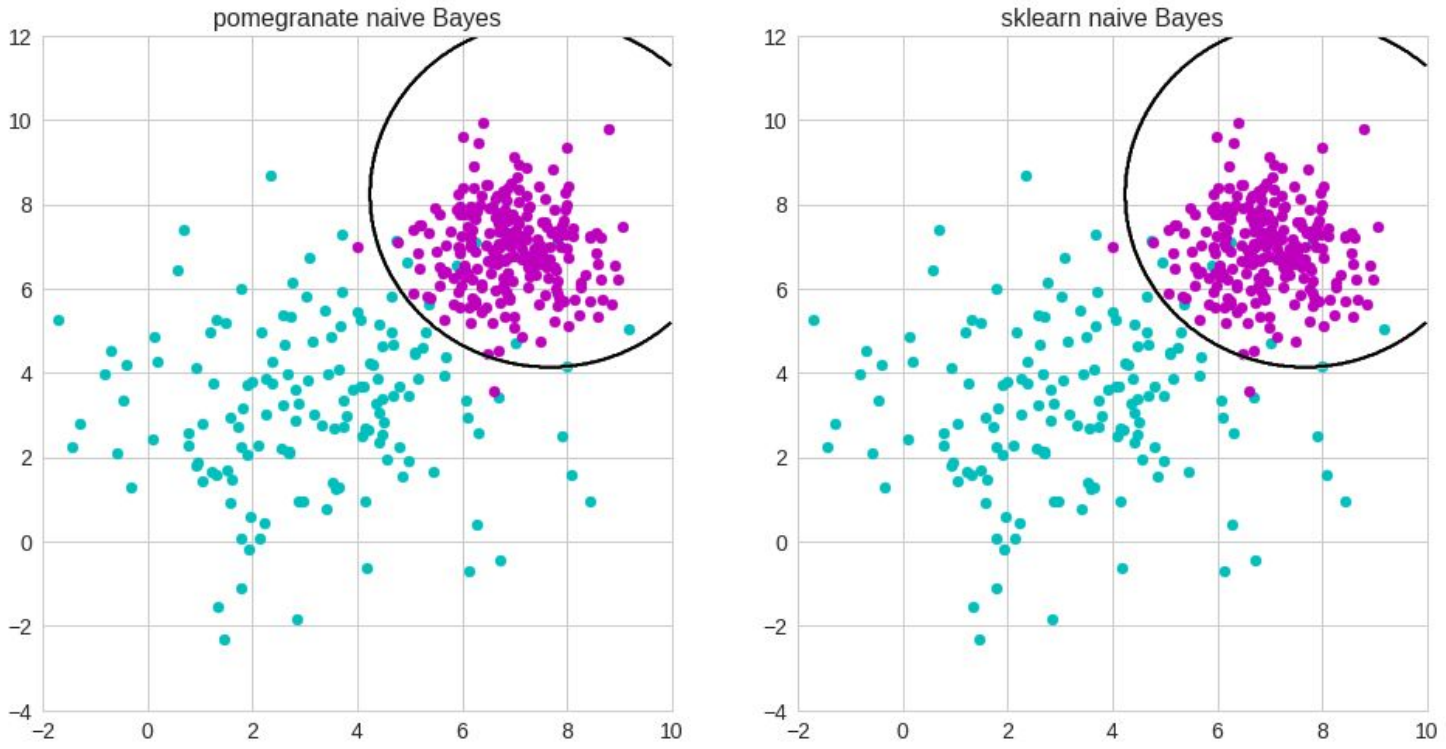
$$Posterior = \frac{Likelihood * Prior}{Normalization}$$

Naive Bayes assumes independent features

$$P(M|D) = \frac{\prod_{i=1}^{d} P(D_i|M)P(M)}{\sum_{M} \prod_{i=1}^{d} P(D_i|M)P(M)}$$

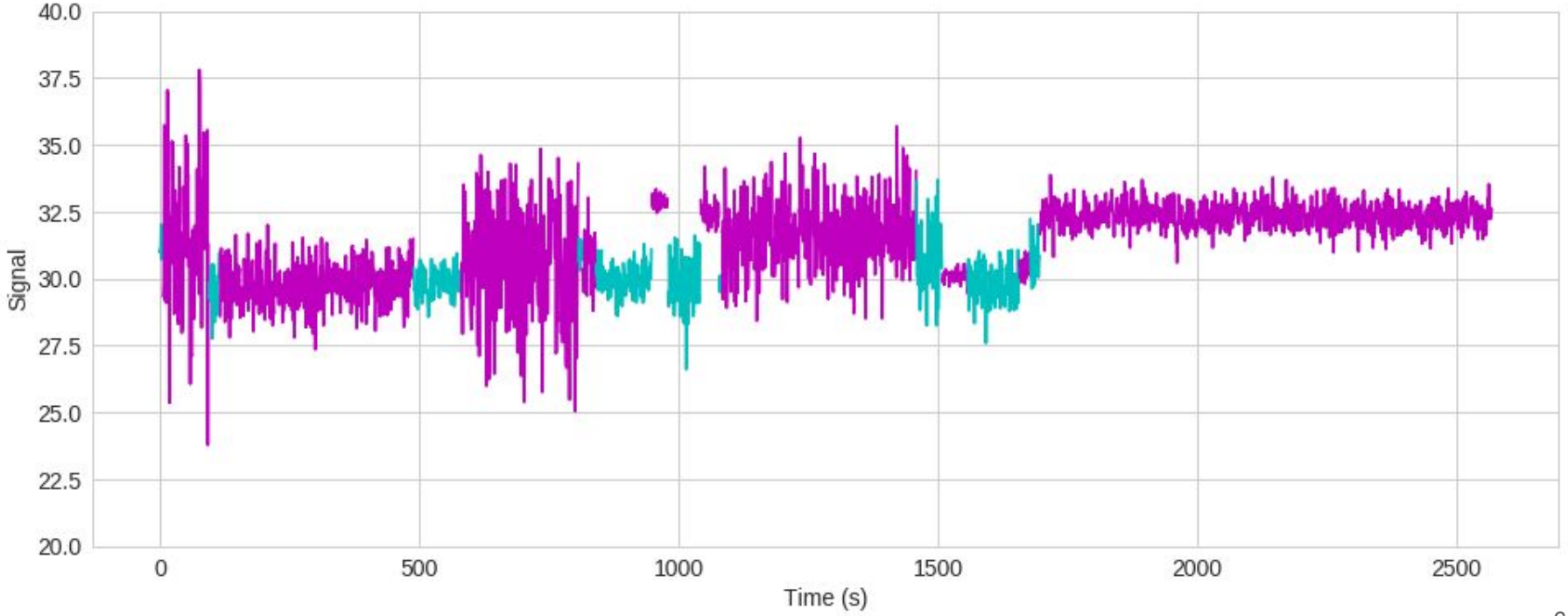
$$Posterior = \frac{Likelihood * Prior}{Normalization}$$

Naive Bayes produces ellipsoid boundaries

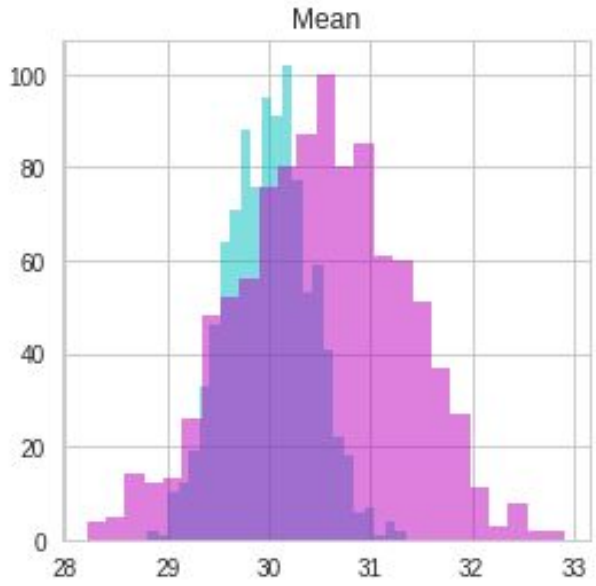


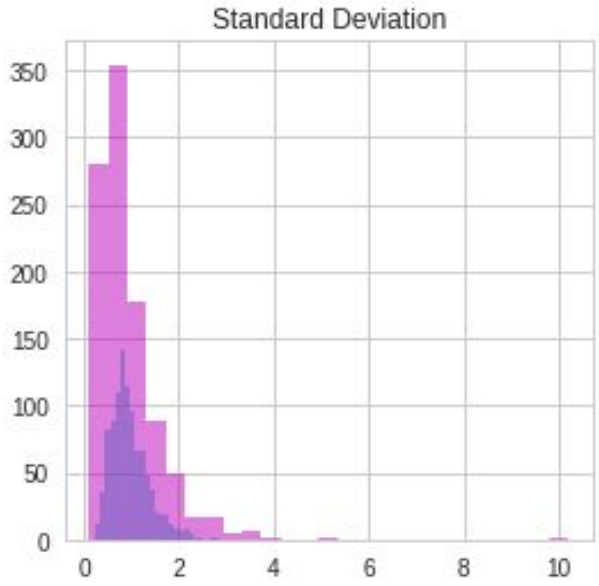
model = NaiveBayes.from_samples(NormalDistribution, X, y)

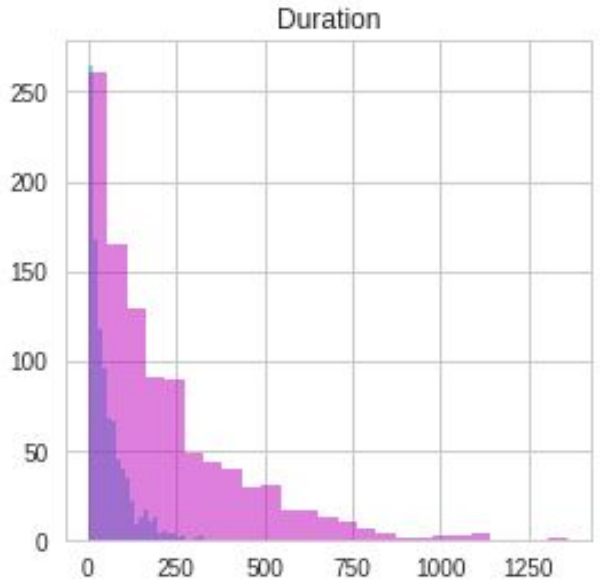
Naive Bayes can be heterogenous



Data can fall under different distributions







Using appropriate distributions is better

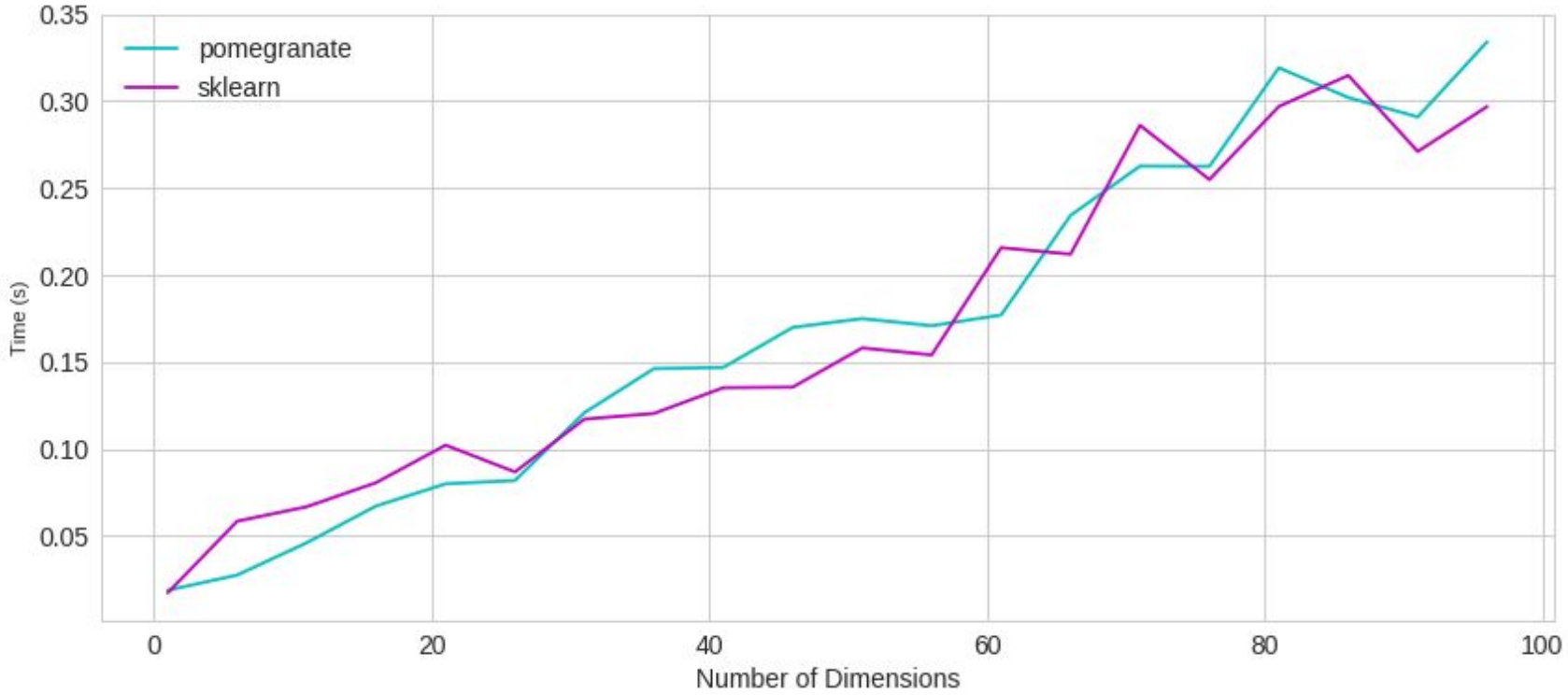
```
model = NaiveBayes.from_samples(NormalDistribution, X_train, y_train)
print "Gaussian Naive Bayes: ", (model.predict(X_test) == y_test).mean()

clf = GaussianNB().fit(X_train, y_train)
print "sklearn Gaussian Naive Bayes: ", (clf.predict(X_test) == y_test).mean()

model = NaiveBayes.from_samples([NormalDistribution, LogNormalDistribution,
ExponentialDistribution], X_train, y_train)
print "Heterogeneous Naive Bayes: ", (model.predict(X_test) == y_test).mean()
```

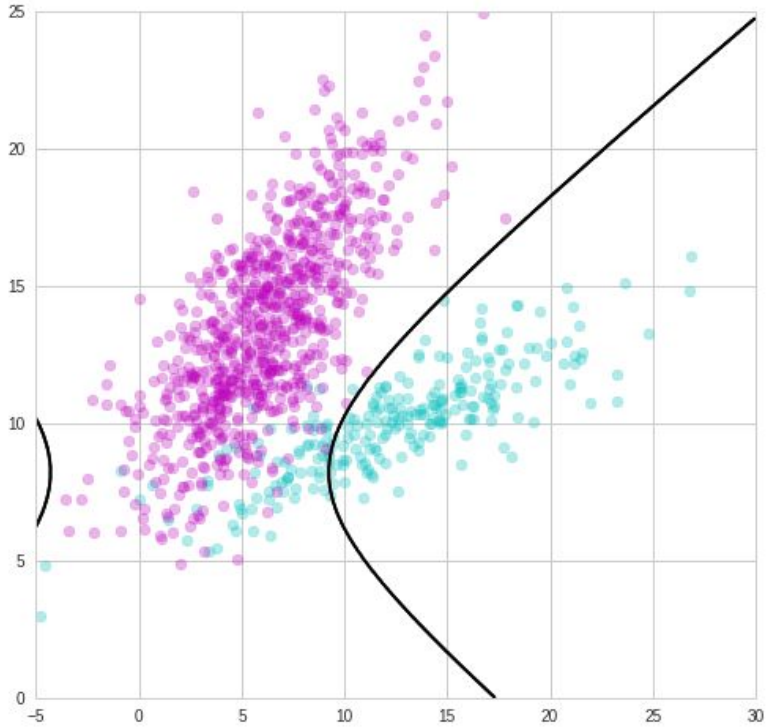
```
Gaussian Naive Bayes:  0.798
sklearn Gaussian Naive Bayes:  0.798
Heterogeneous Naive Bayes:  0.844
```

This additional flexibility is just as fast

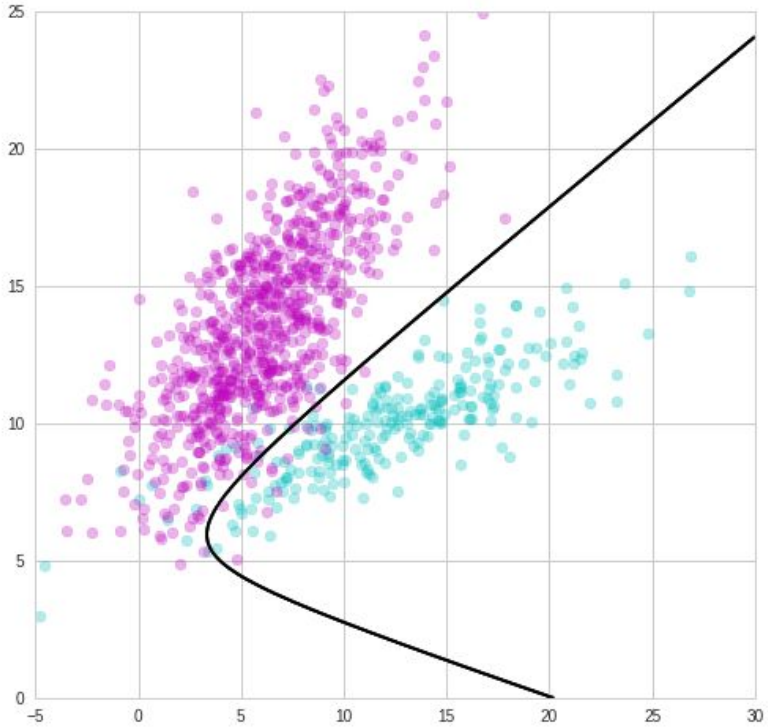


Bayes classifiers don't require independence

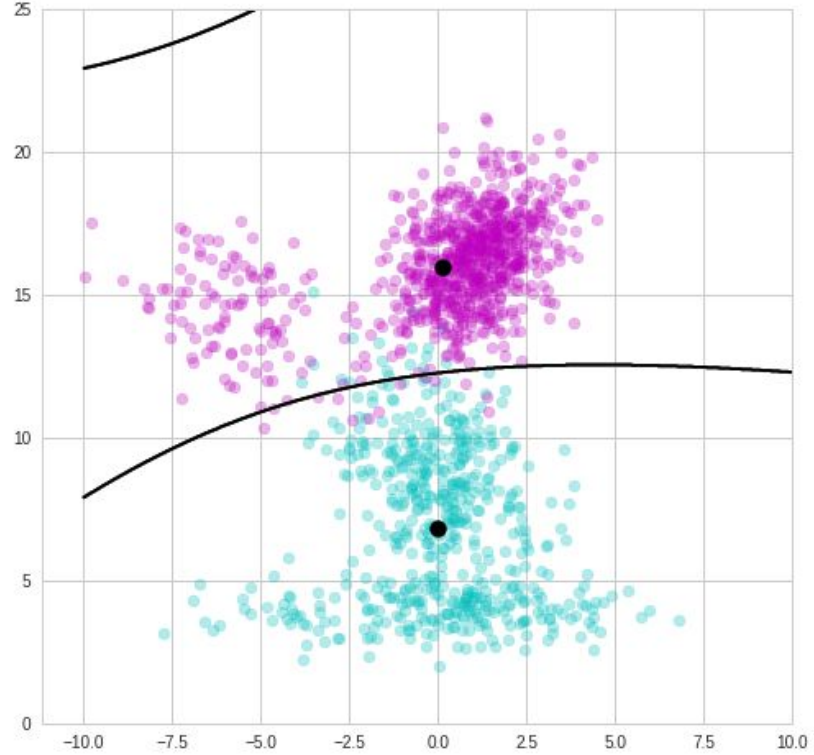
naive accuracy: 0.929

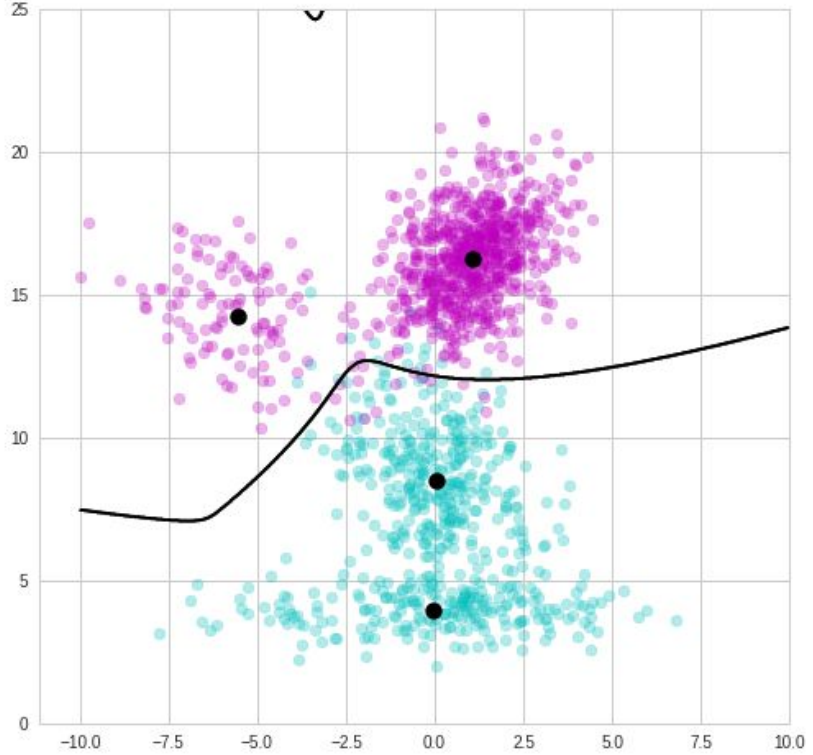


bayes classifier accuracy: 0.966



Gaussian mixture model Bayes classifier





Creating complex Bayes classifiers is easy

```
gmm_a = GeneralMixtureModel.from_samples(MultivariateGaussianDistribution, 2, X[y == 0])
gmm_b = GeneralMixtureModel.from_samples(MultivariateGaussianDistribution, 2, X[y == 1])
model_b = BayesClassifier([gmm_a, gmm_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

Creating complex Bayes classifiers is easy

```
mc_a = MarkovChain.from_samples(X[y == 0])
mc_b = MarkovChain.from_samples(X[y == 1])
model_b = BayesClassifier([mc_a, mc_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

```
hmm_a = HiddenMarkovModel.from_samples(X[y == 0])
hmm_b = HiddenMarkovModel.from_samples(X[y == 1])
model_b = BayesClassifier([hmm_a, hmm_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

```
bn_a = BayesianNetwork.from_samples(X[y == 0])
bn_b = BayesianNetwork.from_samples(X[y == 1])
model_b = BayesClassifier([bn_a, bn_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

Overview: this talk

Overview

Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Finale: Train a mixture of HMMs in parallel

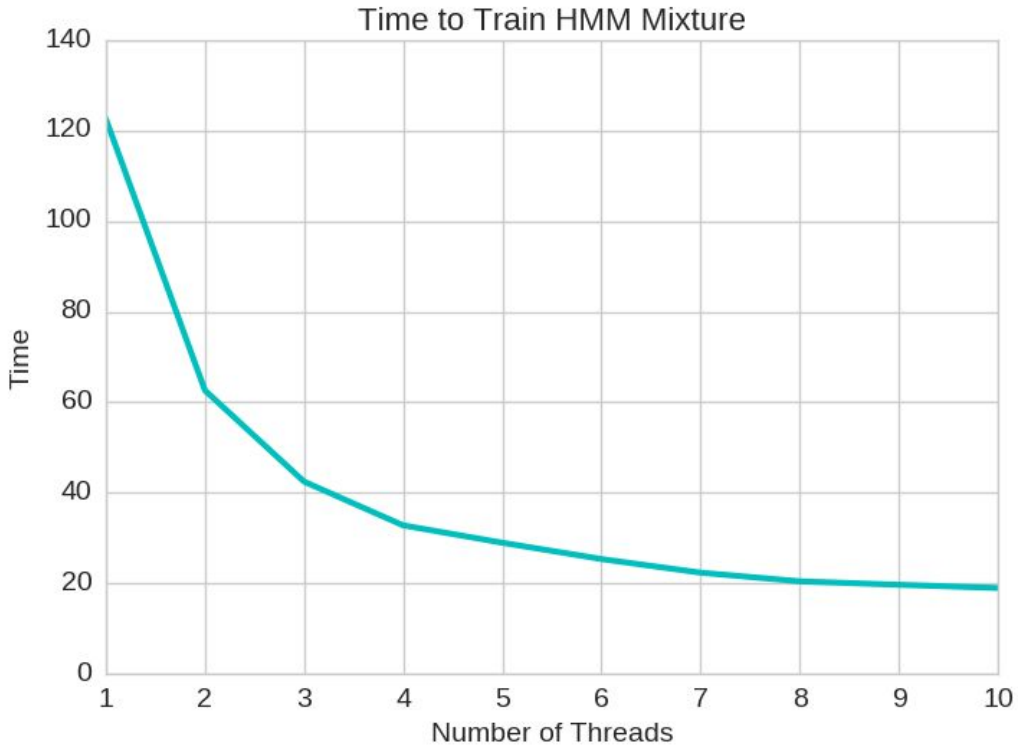
Training a mixture of HMMs in parallel

Creating a mixture of HMMs is just as simple as passing the HMMs into a GMM as if it were any other distribution

```
model_C = create_profile_hmm(dC, I)
model_mC = create_profile_hmm(dmC, I)
model_hmC = create_profile_hmm(dhmC, I)

model = GeneralMixtureModel([model_C, model_mC, model_hmC])
return model
```

Training a mixture of HMMs in parallel



fit(model, X, n_jobs=n)

Overview

pomegranate is more flexible than other packages, faster, is intuitive to use, and can do it all in parallel

Documentation available at Readthedocs

Tutorials available on github

Branch: master ▾ pomegranate / tutorials / Create new file | Upload files | Find file | History

jmschrei ADD bayes backend — Latest commit 724510d 10 hours ago

..		
GGBlasts.xlsx	PyData Chicago 2016	8 months ago
PyData_2016_Chicago_Tutorial.ipynb	FIX markov chain notebooks	3 months ago
README.md	Update README.md	2 years ago
Tutorial_0_pomegranate_overview.ipynb	Minor typos	3 months ago
Tutorial_1_Distributions.ipynb	ENH tutorials	2 years ago
Tutorial_2_General_Mixture_Models.ipynb	FIX hmm dimensionality	11 months ago
Tutorial_3_Hidden_Markov_Models.ipynb	edit tutorial 3 to remove deprecated bake	7 months ago
Tutorial_4_Bayesian_Networks.ipynb	ENH pomegranate vs libpgm tutorial	7 months ago
Tutorial_4b_Bayesian_Network_Structure_Learning.i...	ENH a* search	28 days ago
Tutorial_5_Bayes_Classifiers.ipynb	ADD bayes backend	10 hours ago
Tutorial_6_Markov_Chain.ipynb	FIX markov chain notebooks	3 months ago
Tutorial_7_Parallelization.ipynb	ADD tutorial 7 parallelization	8 months ago

https://github.com/jmschrei/pomegranate/tree/master/tutorials

Thank you for your time.